Ranking for Scalable Information Extraction

Pablo Javier Barrio González

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2015

ABSTRACT

Ranking for Scalable Information Extraction

Pablo Javier Barrio González

Information extraction systems are complex software tools that discover structured information in natural language text. For instance, an information extraction system trained to extract tuples for an *Occurs-in*(*Natural Disaster*, *Location*) relation may extract the tuple ⟨tsunami, Hawaii⟩ from the sentence: *"A tsunami swept the coast of Hawaii."* Having information in structured form enables more sophisticated querying and data mining than what is possible over the natural language text. Unfortunately, information extraction is a time-consuming task. For example, a state-of-the-art information extraction system to extract *Occurs-in* tuples may take up to two hours to process only 1,000 text documents. Since document collections routinely contain millions of documents or more, improving the efficiency and scalability of the information extraction process over these collections is critical. As a significant step towards this goal, this dissertation presents approaches for (i) enabling the deployment of efficient information extraction systems and (ii) scaling the information extraction process to large volumes of text.

To enable the deployment of efficient information extraction systems, we have developed two crucial building blocks for this task. As a first contribution, we have created REEL, a toolkit to easily implement, evaluate, and deploy full-fledged relation extraction systems. REEL, in contrast to existing toolkits, effectively modularizes the key components involved in relation extraction systems and can integrate other long-established text processing and machine learning toolkits. To define a relation extraction system for a new relation and text collection, users only need to specify the desired configuration, which makes REEL a powerful framework for both research and application building. As a second contribution, we have addressed the problem of building representative extraction task-specific document samples from collections, a step often required by approaches for efficient

information extraction. Specifically, we devised fully automatic document sampling techniques for information extraction that can produce better-quality document samples than the state-of-the-art sampling strategies; furthermore, our techniques are substantially more efficient than the existing alternative approaches.

To scale the information extraction process to large volumes of text, we have developed approaches that address the efficiency and scalability of the extraction process by focusing the extraction effort on the collections, documents, and sentences worth processing for a given extraction task. For collections, we have studied both (adaptations of) state-of-the-art approaches for estimating the number of documents in a collection that lead to the extraction of tuples as well as information extraction-specific approaches. Using these estimations we can identify the collections worth processing and ignore the rest, for efficiency. For documents, we have developed an adaptive document ranking approach that relies on learning-to-rank techniques to prioritize the documents that are likely to produce tuples for an extraction task of choice. Our approach revises the (learned) ranking decisions periodically as the extraction process progresses and new characteristics of the useful documents are revealed. Finally, for sentences, we have developed an approach based on the sparse group selection problem that identifies sentences—modeled as groups of n-grams—that best characterize the extraction task. Beyond identifying sentences worth processing, our approach aims at selecting sentences that lead to the extraction of unseen, novel tuples. Our approaches are lightweight and efficient, and dramatically improve the efficiency and scalability of the information extraction process. We can often complete the extraction task by focusing on just a very small fraction of the available text, namely, the text that contains relevant information for the extraction task at hand. Our approaches therefore constitute a substantial step towards efficient and scalable information extraction over large volumes of text.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my advisor, Luis Gravano, for his truly exceptional guidance and relentless patience. Luis has *always* had—often repeated times—the best piece of advice. This, together with his contagious passion for teaching and research, has taught me everything I know today about excellent research. Beyond it all, Luis has been extremely supportive and caring during my years at Columbia, and I will be forever grateful for that.

The final version of this dissertation was possible thanks to my thesis committee: Chris Develder, Kathy McKeown, Ken Ross, and Panos Ipeirotis. They provided insightful comments, which helped me improve the clarity and precision of the final manuscript.

Much of the work in this dissertation is a result of collaboration. The work in Chapter 3 and Chapter 6 would not have been possible without Gonçalo Simões and his advisor, Helena Galhardas. In particular, Gonçalo and I made, I believe, a great team. Our many times heated discussions about our joint work as well as about futebol (Portuguese) or fútbol (Spanish) lead to the best outcome: The result of most of our discussions is part of this dissertation but, more importantly, I became a big fan of F.C. Porto, his team in Portugal, and (I want to believe) he is slowly growing into San Lorenzo, my team in Argentina. The work in Chapter 5 is joint work with Chris Develder, with whom I have been extremely lucky to collaborate. His proactiveness and diligence are admirable, and I hope I acquired at least a bit of such desirable qualities. Finally, the work in Chapter 7 was possible thanks to Anju Kambadur from Bloomberg L.P. Among other things, Anju helped me better interpret many machine learning algorithms from an algebraic standpoint. Looking back, I can say that collaborating with great people not only leads to high-quality research but also gives you formidable friends.

I also want to thank the members of the DB-group for their invaluable feedback and support. In particular, Ken Ross helped me think beyond my research problems. Ken iden-

tified striking analogies to other real-world problems, which many times lead to improving my approaches. Hila Becker, Eva Sitaridi, Ioannis Paparrizos, Orestis Polychroniou, Fotis Psallidas, and Wangda Zhang were always available for discussion, dry runs, coffee breaks, and to lend a hand when things were not going the way they were supposed to. I am walking out with a bunch of great friends that I hope, in one way or another, will be part of more stories.

During my Ph.D. I also had the pleasure to work with many other talented researchers, which I would like to thank. The members of the DETAiLS project taught me what it is like to work in a large academic project. In particular, Kathy McKeown showed me the keys to successful leadership, placing the contentment of the group before it all and guiding the group to a state of steady progress. I also interned at Microsoft Research New York, where I worked with Jake Hofman and Dan Goldstein from the Computational Social Science group. I am thankful to Jake and Dan for letting me contribute to their fascinating, world-changing ideas. The members of the Knowledge Engineering team at Bloomberg L.P., which I visited regularly during my last year as a Ph.D. student, are also a big part of my student life. In particular, James Hodson and Stefano Pacifico welcomed to their group, and always provided invaluable feedback for my research. I hope to keep collaborating with all these researchers in the future.

I would have never finished—or started—the Ph.D. without the unconditional support of my friends, I must admit. My friends from Olavarría, my hometown, have been there all along, rooting for me in every decision I have ever made. My friends from Tandil, my college town, and from Buenos Aires, where I spent every summer during college, told me to be ambitious; they knew, even before I did, that I was going to pursue a Ph.D. New York gave me exceptional friends as well. I met the most brilliant people, many of whom I truly admire, in the Computer Science department. I also made excellent friends during my Friday soccer games (very, very early in the morning). I have to say that soccer has been a rather therapeutic hobby during some difficult days as a Ph.D. student. I cannot close this paragraph without thanking the Argentinian "clan" in New York: I am very lucky to be surrounded by such smart, talented people.

My last words are dedicated to my families—both in Argentina and the U.S.A. Ma y

Pa: El sacrificio que han hecho todos estos años para que no nos falte nada sigue dando sus frutos. Muchísimas gracias por enseñarme a valorar la educación y por darme *todas* las herramientas para cumplir mis sueños. Esta tesis es, por sobre todo, de ustedes. Mati y Luci: Sigan dándome razones para admirarlos cada día más. Yo seguiré insistiendo que el mundo sería un lugar mejor si todos tuvieran la fortuna de tener hermanos como ustedes. Gracias Meli (y Mati), también, por hacerme tío de Emilia, quien me alegra cada día con sus travesuras. Tíos: Gracias por ser mis eternos cómplices y por acompañarme en cada emprendimiento. Cata, Felix, Nelly, y Toto: Quiero que sepan que soy el nieto más afortunado del mundo. De ustedes aprendí, entre muchas otras cosas, que en la vida nada es imposible pero a veces se necesita un esfuerzo extra. Esta tesis es también de ustedes. I was fortunate enough to also have an "adoptive" family in the U.S.A., which often reminded me of the so-called home-feeling, specially, during holiday season. Thank you, Allie, for letting me into your family but, most importantly, for being supportive, caring, and understanding during all these years. My life would not be the same without you.

A Jorge, Carmelo, y Juan

Chapter 1

Introduction

Information extraction systems are complex software tools that discover structured information in natural language text. For instance, an information extraction system trained to extract tuples for an *Occurs-in*(*Natural Disaster*, *Location*) relation may extract the tuple ⟨tsunami, Hawaii⟩ from the sentence: *"A tsunami swept the coast of Hawaii."* Having information in structured form enables more sophisticated querying and data mining than what is possible over the natural language text. Unfortunately, information extraction is a time-consuming task. For example, a state-of-the-art information extraction system to extract *Occurs-in* tuples may take up to two hours to process only 1,000 text documents. Since document collections routinely contain millions of documents or more, improving the efficiency and scalability of the information extraction process over these collections is critical, even over highly parallel computation environments. In this dissertation, we present approaches that address this problem at three granularity levels of the text data: (i) *collection* level, to decide the text collections to process; (ii) *document* level, to decide the text documents to process within these collections; and (iii) *sentence* level, to decide the sentences to process within these documents.

To better illustrate the scalability- and efficiency-related problem above, consider the following example:

Example 1 *An environmental scientist needs to find the safest and most accessible locations in the U.S. to establish evacuation centers for eventual natural disasters. The scien-*

tist decides to collect the natural disasters and affected geographical areas from an extensive group of news collections on the Web that cover most U.S. locations. Intuitively, the scientist could run an information extraction system properly trained to extract tuples for this task over all documents in the collected collections. Unfortunately, such an exhaustive process may take several days—or even months—to finish, which is undesirable due to the intrinsic urgency of the project.

Interestingly, the exhaustive processing of Example 1 above is often unnecessary: Many times only a few text collections—and a few documents within them—produce tuples for a given extraction task. In fact, most relations are topic-specific, in that they are associated mainly with documents about certain topics. For example, less than 2% of the 1.03 million documents in collections 1-5 from the TREC conference [TRE00] produce *Occurs-in* tuples when processed with a state-of-the-art information extraction system and, not surprisingly, most of these documents are about environment-related topics. Moreover, within these documents, only 3.5% of the sentences include mentions of tuples for this extraction task. If we could identify the collections, the documents in these collections, and the sentences within these documents that lead to the extraction of tuples, we would be able to successfully complete the extraction task while decreasing the extraction time dramatically.

Identifying the collections, documents, and sentences that are worth processing is a challenging proposition for multiple reasons. Specifically, this identification step has to be: (i) efficient, to avoid becoming computationally more expensive than an alternative exhaustive processing; (ii) high-precision, to avoid processing collections, documents, and sentences that will not lead to the extraction of tuples; and (iii) high-recall, to avoid missing valuable collections, documents, and sentences for the extraction task of choice. As we will see, this dissertation advocates efficient ranking approaches at all relevant granularities of the data to effectively address these challenges.

Earlier efforts to identify collections for an extraction task (e.g., [AC05; JS09]) have focused on the quality of the extraction output, rather than its volume. Specifically, these approaches aim at devoting the extraction effort to collections that are likely to produce high-quality extraction output (e.g., tuples extracted with high confidence) and ignore the rest, for efficiency. These approaches are motivated by the fact that running an information

extraction system over collections with large numbers of low-quality tuples incurs considerable overhead, because the extracted tuples are likely to be disregarded. Although the quality of the extraction output is indeed important, the complementary problem of effectively computing the number of tuples that we can potentially extract from the documents in a collection, which has been largely ignored by these approaches, is also of critical importance: For collections with comparable extraction output quality, we would like many times to focus the extraction effort on collections with large numbers of tuples, namely, the *useful* collections for the extraction task. We address this important problem in Chapter 5.

In a similar vein, existing approaches to improve the efficiency and scalability of information extraction systems over large text collections (e.g., QXtract [AG03], PRDualRank [FC11], and FactCrawl [BLNP11a]) have addressed the document-level problem introduced above, but exhibit crucial limitations. Specifically, these approaches are based on the observation that the documents that produce tuples for an extraction task of interest, namely, the *useful documents*, tend to share certain distinctive words and phrases. For example, documents containing mentions of earthquakes—hence useful for the *Occurs-in* relation—many times include words like "richter" or "hypocenter." These words and phrases can then be used as keyword queries, to retrieve from the collection the (hopefully useful) documents that the extraction system will then process. To discover these words and phrases, a critical step in the process, these techniques analyze a sample of documents from the collection of interest, trying to keep the size of this document sample small, so that the overhead of the querying process remains at reasonable levels. Because the samples are small, unfortunately, these approaches often compromise the precision and recall of the extraction process. In Chapter 6, we propose an approach that effectively addresses these precision and recall limitations of existing approaches.

Beyond the document-level strategies above, other approaches (e.g., [NVB01; XGZ11; WSE13]) have tackled the efficiency of the extraction process at a finer granularity of the text. Specifically, given a set of input documents, the approach in [NVB01] uses classifiers to filter the sentences that are likely to produce tuples for the extraction task at hand. (The approach in [WSE13] includes this filtering step as a component of the relation extraction system.) Unfortunately, such filtering steps are far from perfect and often miss

useful sentences for the extraction task, or sentences that produce tuples for the extraction task at hand. The approach in [XGZ11] proposed instead to rank portions of text documents (e.g., sentences, paragraphs, or fixed- or variable-size text windows), which are often referred to as *passages*, for the (related) task of efficiently extracting attributes of entities (e.g., people or organizations) from text. Following a strategy similar in spirit to that of the document-level techniques above, the approach in [XGZ11] learns keyword queries that (potentially) retrieve passages with mentions of tuples for the extraction task of choice. Finally, this approach retrieves and ranks the passages for extraction via *passage retrieval* [KZ01], a family of techniques that identify passages that are topically relevant to a given query. Unfortunately, this ranking step often involves detecting entities of interest in the passages, which is often time-consuming and conflicts with our efficiency requirement above. Moreover, this approach ignores the valuable information that is observed along the extraction process and, in effect, may process passages that lead to the extraction of already seen tuples. In Chapter 7, we propose an approach that effectively addresses these limitations and prioritizes *novel sentences*, or useful sentences that produce unseen, novel tuples.

To enable solving the problems above at scale, two lower-level problems also need to be addressed. We illustrate these problems in the next example:

Example 1 (continued) *Say the environmental scientist in Example 1 has identified a rather comprehensive list of U.S. news websites (e.g., the list of almost 3000 U.S. newspapers at* `http://www.world-newspapers.com/usa.html` *), which provides an ample coverage of the geographical locations of interest. The environmentalist now faces the deployment of the extraction task over these collections, which poses two crucial challenges. First, the scientist needs to develop an information extraction system for the Occurs-in relation, the extraction task of interest. Unfortunately, although many information extraction systems are available, adapting them to this new task and, more importantly, assessing their performance to select the best system is cumbersome. Second, to process these collections efficiently, and in a timely manner, the extraction process needs to gather a representative sample of useful documents from each collection. Unfortunately, existing approaches to collect such document samples often miss relevant groups of useful documents, which in turn impacts the overall performance of the extraction process.*

As shown in Example 1 above, studying and empirically evaluating approaches for collection-, document-, and sentence-level problems above at scale requires addressing two important challenges. *(1) Building Information Extraction Systems:* Information extraction systems are in general difficult to implement, train, and evaluate, because of the many text-processing steps (e.g., word tokenization, part-of-speech tagging, and other more complex feature extraction steps) that are involved during extraction. Even more importantly, these systems are often difficult to adapt to new extraction tasks and new text formats. *(2) Building Representative, Extraction-Specific Document Samples from Text Collections:* As we will see, many of the existing and new approaches for the problems of focus require having a representative document sample from each collection. These samples need to effectively represent the (often rare) useful documents in text collections (e.g., by including document with diverse tuple attributes) and need to be collected in a fully automated fashion (i.e., without human intervention). Moreover, document sampling should occur in a collection-specific way, because the focus and language of each collection generally differs from those of other collections. We will address these two crucial challenges in Chapters 3 and 4.

Specifically, the key contributions of this dissertation are as follows:

- **Toolkit for Building Relation Extraction Systems:** In Chapter 3, we present REEL (**RE**lation **E**xtraction **L**earning framework), an open-source framework to easily develop and evaluate relation extraction systems. REEL provides the code and infrastructure to: (i) handle various input text formats, which enables operations over different text collections; (ii) plug in appropriate text processing steps and tools, which enables diverse processing of the text with minimal effort; (iii) define and combine conceptual relation constraints that are automatically enforced; (iv) decouple learning and extraction from the text processing, which enables the straightforward integration and re-usability of different extraction algorithms; and (v) uniformly execute and evaluate relation extraction systems, which enables the testing and fair assessment of these systems. REEL, in contrast to existing toolkits, effectively modularizes the key components involved in relation extraction systems. To define a relation extraction system for a new relation and text collections, users only need to specify the parsers to

load the collections, the relation and its constraints, and the learning and extraction techniques, which makes REEL a powerful framework to enable the deployment of relation extraction systems for both research and application building.

- **Study of Document Sampling Strategies for Information Extraction:** In Chapter 4, we systematically study the space of query-based document sampling techniques for information extraction. Specifically, we consider (i) alternative query execution schedules, which vary on how they account for the query effectiveness; and (ii) alternative document retrieval and processing schedules, which vary on how they distribute the extraction effort over documents. We conduct a large-scale and fine-grained experimental evaluation over real Web collections, and for a large variety of information extraction tasks, to assess the merits of the alternative query execution and document retrieval and processing strategies. Our conclusions are twofold. Regarding query execution, schedules that focus on queries with a high fraction of useful documents, namely, the *effective* queries, improve sampling efficiency. In contrast, schedules that prioritize less effective queries improve sampling quality, because in this case many (potentially diverse) queries need to be issued to retrieve a desired number of useful documents, hence leading to high-quality document samples. Regarding document retrieval and processing, schedules that process the documents for each query exhaustively at once improve sampling efficiency when the sampling technique focuses on effective queries. In contrast, schedules that process documents incrementally and in rounds improve sampling quality, because a larger variety of documents—from a larger number of queries—is processed.

- **Methods for Ranking Text Collections for Information Extraction:** In Chapter 5, we address the problem of identifying useful collections for an extraction task. We introduce the problem of ranking text collections for efficient and scalable information extraction, and develop lightweight, query-based approaches to estimate the number of useful documents for a given information extraction task in a text collection. Our approaches cover the three classical families of estimation techniques for text collections, namely, surrogate-, query-pool-, and pool-free-based techniques,

which are suitable for different extraction scenarios, as we will see. We compare both (adaptations of) state-of-the-art resource selection strategies, and information extraction-specific approaches on a large-scale experimental evaluation over realistic Web collections, and for several different information extraction tasks. Our results show the merits and limitations of the alternative families of approaches, and provide a roadmap for addressing this critically important building block for efficient, scalable information extraction.

- **Techniques for Ranking Text Documents for Information Extraction:** In Chapter 6, we address the problem of identifying useful documents for an extraction task. We advocate an adaptive document ranking approach that addresses the precision and recall limitations of the state-of-the-art techniques. Specifically, we propose a principled, efficient learning-to-rank approach that prioritizes documents for an information extraction task by combining: (i) online learning [SSSS07], to train and adapt the ranking models incrementally, hence avoiding computationally expensive retrains of the models from scratch; and (ii) in-training feature selection [GE03], to identify a compact, discriminative set of words and phrases from the documents to train ranking models effectively and efficiently. Importantly, our approach revises the document ranking decisions periodically, as the ongoing extraction process reveals (fine-grained) characteristics of the useful documents for the extraction task at hand. Our approach thus manages to capture, progressively and in an adaptive manner, the heterogeneity of language and content typically exhibited by the useful documents, which in turn leads to information extraction executions that are substantially more efficient—and effective—than those with state-of-the-art approaches, as we will see. In summary, we present an end-to-end document ranking approach for effective and efficient information extraction in an adaptive, online, and principled manner.

- **Approach for Ranking Sentences for Information Extraction:** In Chapter 7, we address the problem of identifying useful and novel sentences for an extraction task. Specifically, we propose a principled, efficient approach that exploits a forward greedy sparse group selection strategy [LSA09] to identify the (rare) useful sentences

from a set of documents. Our approach models each sentence as a group of n-grams and iteratively selects the sentence that best explains a carefully designed representation of the extraction task at hand. We build this representation of the extraction task gradually, as the extraction process progresses, to capture all relevant aspects of the task. During sentence ranking, our approach updates this representation to account for the relevant aspects of the extraction task that have been already explained by other previously selected sentences. By doing this, our approach manages to prioritize sentences that lead to the extraction of unseen tuples. Furthermore, our approach provides for trading relevance and novelty in a robust manner, to suit different application requirements. Our experimental evaluation over a broad range of extraction tasks shows the merits and limitations of all relevant building blocks in our approach and, more importantly, shows the significant efficiency improvements that can be obtained by effectively prioritizing sentences.

To illustrate the challenges of running an information extraction system over large text collections, necessary for this dissertation, we provide the relevant background in Chapter 2. We describe related work in Chapter 8, and then present our conclusions and discuss directions for future work in Chapters 9 and 10, respectively.

Chapter 2

Background

This chapter provides necessary background and defines the high-level problem that we study in this dissertation. Specifically, Section 2.1 describes information extraction, the text-centric task on which we focus in this dissertation. Section 2.2 discusses the types of text collections that we study, along with their associated challenges. In turn, Section 2.3 defines the problem of information extraction over large text collection, a task of critical importance and the focus of this dissertation. Finally, Section 2.4 reviews applications empowered by performing information extraction over large text collections, and that in effect greatly benefit from the contributions in this dissertation.

2.1 Information Extraction

Natural language text often embeds valuable structured information, which typically consists of entities, attributes, or relations between them. For example, the sentence *"A tsunami swept the coast of Hawaii."* includes two entities, namely, *tsunami* and *Hawaii*, and a semantic relation between them, namely, that Hawaii was affected by a tsunami. *Information extraction* refers to the automatic identification and extraction of the rich structured information from text [Sar08]. This structured information is much better suited to sophisticated querying and analysis than the unstructured natural language text. In this dissertation, we focus on *relation extraction*. We now first define the relation extraction problem (Section 2.1.1). We then describe how typical relation extraction systems operate over text

(Section 2.1.2). Finally, we discuss certain efficiency-related properties that make relation extraction systems particularly challenging to be deployed at scale (Section 2.1.3).

2.1.1 The Relation Extraction Problem

Many data-centric tasks require identifying semantic relations between entities from text, as finding entities in isolation may be insufficient. For example, we may need to identify from a news article what geographic location has been affected by a natural disaster or what person has been accused of a crime and when. *Relation extraction* refers to the automatic detection of relations between two or more entities from text. For our examples, a relation extraction system properly trained to extract tuples for an *Occurs-in*(*Natural Disaster*, *Location*) relation may extract the tuple ⟨tsunami, Hawaii⟩ from the sentence: *"A tsunami swept the coast of Hawaii."* Likewise, another system properly trained to extract tuples for a *Charged*(*Person*, *Charge*, *Date*) relation may extract the tuple ⟨Mark Chapman, second-degree murder, 1981⟩ from the text excerpt: *"John Lennon's killer, Mark Chapman, was sentenced in 1981 to 20 years to life in prison after pleading guilty to second-degree murder."*

The relation extraction problem gained initial attention during research competitions for specific domains and was later extended to other areas as well. One of the first competitions to promote relation extraction was the Message Understanding Conference (MUC), specifically, in its sixth [GS96] and seventh [Chi98] editions. The goal was to extract relations such as *Employee-of*(*Person*, *Organization*) or *Location-of*(*Organization*, *Location*) from news articles. Other competitions were the Automatic Content Extraction (ACE) task [DMP+], which included multiple relations between *person*, *organization*, *facility*, *location*, and *geo-political entities*, also over news articles, and the BioCreAtIvE II Protein-Protein Interaction tasks [HYBV05; TAC06] in bioinformatics over scientific literature. Beyond news and scientific articles, the relation extraction problem has been studied over a wide variety of text sources (e.g., emails [JKR+06], Wikipedia [SIW06; SKW07], the general Web [AG00; EBSW08]). We summarize next how relation extraction systems operate over natural language text.

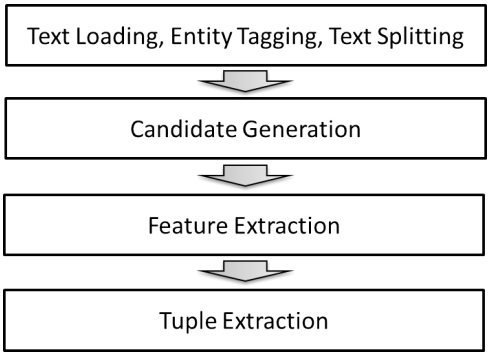


Figure 2.1: Key steps in relation extraction.

2.1.2 Extracting Relations from Text

To discover relations in a text document, a relation extraction system needs to perform a series of steps, which are illustrated in Figure 2.1. The relation extraction system starts by loading the contents of a given text document, tagging the entities of interest, and splitting the tagged text into *text segments* (see Text Loading, Entity Tagging, and Text Splitting step in Figure 2.1). Text segments are often sentences, but they can also be paragraphs, the entire document text, or combinations thereof. For each segment, the system then generates zero or more *candidate text segments*, namely, the text segments that satisfy all relation and entity constraints for the relation at hand (e.g., entities need to be of a certain type, say, *Natural Disaster* or *Location*, or entities need to be mentioned within N words of each other), and thus contain a mention of a potential tuple (see Candidate Generation step in Figure 2.1). Then, the system extracts other relevant features (e.g., sequences of characters in the context of the entities, distance between entities, shallow parse trees, dependency graphs), which are often task-specific (see Feature Extraction step in Figure 2.1). The final step consists of using these extracted features as input to the tuple extraction algorithm (e.g., [BM05b; BM05a; CS04; FSE11; GJJM05; Kam04; NWS12; ZAR+03; ZG05]), which often relies on a binary classification approach: Based on decisions learned during a training step, the tuple extraction algorithm labels a candidate text segment as positive—thus concluding that the entities in the candidate are related—or negative otherwise (see Tuple Extraction step in Figure 2.1).

The tuple extraction algorithm discussed above can be classified into three broad classes,

namely, pattern-, feature-, and kernel-based systems, according to how they model relations. Pattern-based strategies [AG00; Bri99; FC11; NWS12] aim at identifying text and grammatical patterns that signal a relation of interest between entities. For instance, the occurrence of the text pattern "$\langle Person \rangle$ was arrested for $\langle Charge \rangle$ on $\langle Date \rangle$" in a given text excerpt may determine the existence of the above *Charged*(*Person*, *Charge*, *Date*) relation. Feature-based relation extraction systems (e.g., [FSE11; GJJM05; Kam04; ZG05]), on the other hand, operate over a predefined feature space (e.g., consisting of syntactic features, such as the number of words between entities, semantic features, such as the path between entities in the dependency graph, and lexical features, such as the words between entities) and aim at learning which features—in this feature space—are discriminative of the relation of interest. For example, these systems can learn that the occurrence of words like "arrested" or "trial" anywhere in a given text excerpt are important indicators of the *Charged* relation. Finally, kernel-based relation extraction systems (e.g., [BM05b; BM05a; CS04; ZAR+03]) exploit kernel functions to explore a large, not explicitly predefined, feature space that feature-based approaches are unable to handle. Kernel-based systems can learn that, for instance, certain shapes of shallow parse trees are more likely to include related entities than others (e.g., if they contain multiple subtrees that cover the related entities and that share multiple nodes with parse trees of other manually annotated text excerpts).

Because of all these operations that are typically involved, relation extraction is a computationally expensive process. Next, we analyze the efficiency of these typical operations.

2.1.3 Efficiency of Relation Extraction Systems

The efficiency of a relation extraction system mainly depends on the efficiency of its multiple operations, as argued above. Table 2.1 shows per-sentence, single-threaded time measurements of a selection of natural language processing (NLP) tasks that are often required to run a full-fledged relation extraction system over text documents. These estimates are based on methods several times more efficient than those generally deployed in information extraction systems, and that often exhibit close to state-of-the-art accuracy for their corresponding task. Each NLP task in the table depends on all—or a subset of—the tasks shown

Task	Time per sentence (ms)	Toolkit or algorithm	Source	Performed over
Sentence splitting	0.1	PTB	[AS12]	**All documents**
Tokenization	0.1	PTB	[AS12]	
Part-of-speech tagging	7.4	ClearNLP	[Cho12]	
Shallow parsing	42	Search Algorithm	[TT05]	
Dependency parsing	25.6	ClearNLP	[Cho12]	
Semantic role labeling	8.4	ClearNLP	[Cho12]	
Named entity recognition	1.1	SENNA	[AS12]	
(per entity)	428.5	Nested	[FM09]	
Coreference resolution	169	CROCS	[DW15]	**Documents with entities**
Relation Extraction	766	Tree Kernel	[dSMSB13]	
	67	OLLIE		

Table 2.1: Efficiency of various NLP tasks commonly used to support relation extraction. Measurements are obtain from single-threaded executions, as reported in the source reference.

higher in the table, and thus have to be computed sequentially for a given sentence.

Based on these shown estimates, and ignoring the time incurred to fetch a sentence, we could decide whether a sentence within a document is a candidate (i.e., the sentence includes the entities of interest) or not in approximately 90 ms (see task performed over all documents in Table 2.1). This value may of course increase considerably if more sophisticated named entity recognition algorithms (e.g., [FM09]) are needed for the entities in the relation of choice. In turn, the detection of all entities of interest in the document triggers the execution of other time-consuming tasks, such as coreference resolution and relation extraction per se (see Documents with entities entries in Table 2.1). Coreference resolution, which is included in the entity tagging task in Section 2.1.2, is a time-consuming process often performed to detect all instances of the entities of interest. Finally, once all entities have been detected, the relation extraction system runs the relation extraction algorithm over each candidate sentence. This step may incur considerable time, because a single sentence can produce multiple candidate sentences, as discussed.[1]

Besides the NLP tasks above, relation [AS12] extraction systems often need additional "clues" in the form of features, some of which require expensive feature generation steps. Specifically, these features can be simple features (e.g., a Boolean value indicating whether

[1]Some of the NLP tasks in Table 2.1, namely, the tasks from sentence splitting through semantic role labeling, could be computed in advance, since they are often shared across different extraction tasks. However, these tasks comprise only a small fraction of the overall extraction time, as shown. More importantly, running these tasks exhaustively over the available text would be unnecessary because, as we will see, often only a small portion of the available text leads to the extraction of tuples.

a word is capitalized), which can be computed relatively efficiently, or rather more complex features (e.g., confidence of a string matching algorithm over entries in a remote database of product names), which may require more computationally involved tasks [Sar08]. Moreover, and similarly to the NLP tasks above, some of the features on which relation extraction algorithms rely (e.g., the distance between nodes in a dependency graph [BM05a]) depend on the values of other features or output NLP tasks and can only be computed sequentially. Generating these features is thus also an important efficiency constituent of relation extraction systems.

Based on our discussion above, running a full-fledged relation extraction system over a sentence is a time-consuming process. Because we often need to run relation extraction systems over large text collections, as we will see, addressing the efficiency and scalability of the extraction process, the focus of this dissertation, is of critical importance.

2.2 Text Collections

Our goal in this dissertation is to scale the execution of information extraction systems to large text collections. We now define—and discuss the challenges associated with—the two families of text collections, namely, *fully-accessible* (Section 2.2.1) and *deep web* (Section 2.2.2) text collections, that we consider throughout this dissertation.

2.2.1 Fully-Accessible Text Collections

The first family of text collections that we consider, namely, the *fully-accessible* text collections, consists of the collections that provide unrestricted access to their documents. Collections in this family comprise locally available archives, such as company emails (e.g., [EGN14]), human- and machine-generated documents (e.g., [D4D14]), news archives (e.g., [San08]), or scientific publications (e.g., [Els15]), and crawls of Web text collections, such as news websites (e.g., [CNN15]), blogs (e.g., [ICW11]), Wikipedia [Wik15], or a combination thereof (e.g., [Clu09; Clu12]) that can obtained through standard Web crawlers [ON10].

Text documents in these (fully-accessible) collections above many times include valuable structured information that can be particularly helpful for various text-centric tasks. This

information is rather specific of each type of collection and using this information would not generalize well to other types of collections. For example, scientific articles typically include metadata (e.g., authors, affiliations, keywords, category) that has proven useful for citation prediction [YHO+11]. Likewise, emails, blogs, and news articles often include a creation date, which is useful for the detection and tracking of events mentioned in the documents [BNG10]. News articles and Wikipedia documents many times include links to other documents in the collection—or to other collections—and to knowledge bases, which enrich the contents of the document for tasks like named entity disambiguation [HOD12]. In this dissertation, we adopt a rather general approach: As we will see, our methods rely only on the text contents, a shared characteristic across all collections.

2.2.2 Deep Web Text Collections

However, not all the contents of the Web are reachable through standard crawlers [ON10]. In fact, there is content "hidden" behind web search interfaces, in what is often referred to as *the deep web* [MAAH09; MKK+08; ZWC+13]. The second family of text collections that we consider, namely, the *deep web* text collections, consists of the collections in the deep web that contain natural language text documents.

As an example of a deep web text collection, consider the Federal Emergency Management Agency (FEMA) collection [FEM15]. The FEMA collection is one of the most up-to-date resources for natural disasters and other hazards in the United States. Users can find information about disasters, their consequences as well as the government reaction to them. Another example of deep web text collections is PubMed [Pub15], a well-known collection for life sciences and biomedical research that hosts over 22 million abstracts and references in the medical field. Biomedical scientists can search for full articles, similar papers, and their references on a large variety of biomedical topics. In addition to FEMA and PubMed, the deep web is home of a large number of high-quality collections across many domains [Zil08]. There is, in effect, a wide range of text-centric tasks that can benefit from gathering and exploiting the valuable information buried in these collections. Furthermore, the collective content in deep web collections may exceed in volume, according to some estimates, that of the crawlable, or "surface" web [HPZC07;

MJC+07].

Accessing the contents of deep web collections poses several challenges. These challenges are principally related to the querying interface that each collection provides [MKK+08] and to the document indexing and retrieval criteria that different collections adopt [BYG11]. Querying interfaces across deep web collections, on the one hand, are rather heterogeneous, in that they vary in the input methods for queries (e.g., GET or POST methods) and in their navigation across returned documents (e.g., allow random access to all retrieved documents or provide navigation links, such as "more results" or "next", only). We therefore need fully automatic techniques to effectively query and navigate the contents of the vast number of deep web collections on the Web. We describe the end-to-end, fully automatic system that we developed for this critical task in Section 4.3.

Other challenges of deep web collection are associated with the indexing and retrieval criteria that each collection adopts [BYG11], as mentioned above. Notably, these criteria can affect considerably the documents that are retrieved for a given query. During indexing, different collections can vary, among others: (i) indexing depth, which refers to the maximum number of tokens (e.g., words and phrases) that are indexed per document; (ii) parsing and tokenization, which refers to the algorithm adopted to identify terms in the documents; and (iii) indexing scope, which refers to the sources (e.g., anchor text or document text) from which the indexing algorithm obtains tokens for indexing. During retrieval, deep web collections can differ along important dimensions, including: (i) query result limit, which refers to the maximum number of documents than can be retrieved for an input query; and (ii) ranking strategy, which refers to how documents are prioritized (e.g., by relevance to the query, by diversity, by freshness, or a combination thereof). Different variants of these components may cause queries to retrieve documents that do not include the words in the query (e.g., because these words only appear as anchor text in other documents) and prevent other queries from retrieving documents that include the words in the query (e.g., because the results of a query reached the query result limit). Addressing these challenges is crucial to derive reliable, unbiased statistics from the deep web collections [ZZD11; BYG11; ZZD13].

2.3 Information Extraction over Large Text Collections

We now discuss the task of *information extraction over large text collections*, which aims at exploiting the wealth of valuable data in the collections. Specifically, we first describe a state-of-the-art approach for deploying an information extraction system over available large text collections, which we illustrate in Figure 2.2. We then discuss multiple opportunities to improve this process substantially and that are the basis of this dissertation, as we will see.

The extraction process starts by retrieving a sample of useful and useless documents from the collections (see Sample Generator in Figure 2.2). This document sample needs to be collected efficiently and should effectively represent the diversity of useful documents in the collections. An existing approach to collect useful documents for this document sample is to turn tuples for the relation of interest (e.g., ⟨tsunami, japan⟩, ⟨earthquake, san francisco⟩) into queries [AG03], in a bootstrapping fashion. Specifically, starting with a small set of "seed" tuples, this approach iteratively builds queries from the tuples that the information extraction system extracts from retrieved documents.

After collecting a document sample from each collection, the extraction process automatically annotates the sample by running the information extraction system over the retrieved documents. These (now-annotated) documents are in turn used to generate queries that retrieve potentially useful documents for the extraction task at hand (see Automatic Query Generator and Dispatcher in Figure 2.2). For instance, the extraction process can learn queries such as [richter], [earthquake], and [aftermath] to retrieve documents about earthquakes from the earthquake.usgs.gov collection, and other queries such as [storm], [damage], and [injured] to retrieve other disasters from other collections.

Once the extraction process has issued the learned queries to the collections, the information extraction system extracts *Occurs-in* tuples from the retrieved documents (see Extraction Plan Executor in Figure 2.2) by performing the discussed steps in Figure 2.1 over each retrieved document. During this step, the extraction process could: (i) rank the documents to process according to their usefulness to the extraction task at hand (e.g., by using FactCrawl [BLNP11a]), to prioritize the extraction effort and improve the efficiency of the process; or (ii) filter sentences or paragraphs of the document contents (e.g., by using

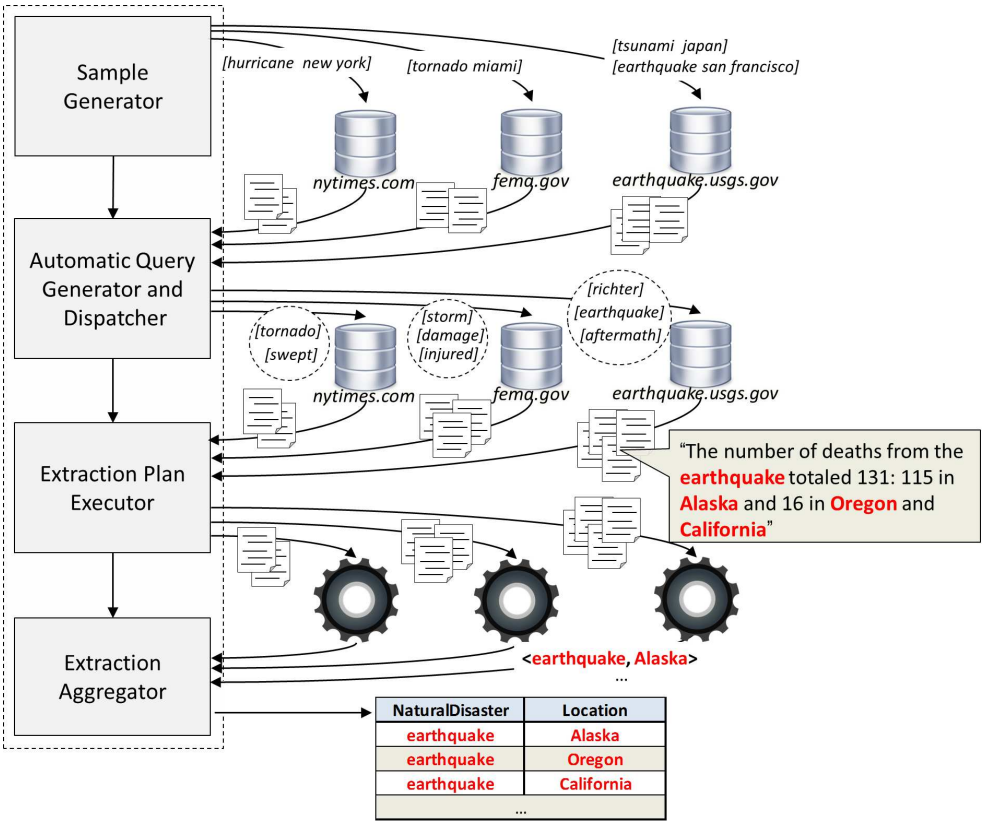


Figure 2.2: Extracting *Occurs-in* tuples from text collections.

the approach in [NVB01]) that are unlikely to be useful for the extraction task.

The final step of the extraction process involves the population of the *Occurs-in* table using the extracted tuples from all processed text collections (see Extraction Aggregator in Figure 2.2), which is valuable for multiple applications, as we will see. During this final stage, the extraction process prepares the tuples to be appropriately reported as output. For instance, we can normalize tuples by disambiguating their corresponding entities [Win99] to, in turn, generate an elegant, easy-to-analyze output.

We identify multiple opportunities for improving the efficiency and scalability of the extraction process above. Our first observation is on the document sampling strategies that initiate the overall extraction process, and that are crucial for the overall performance of the extraction process: Earlier efforts to address the efficiency and scalability of the extraction process (e.g., QXtract [AG03], FactCrawl [BLNP11a], and PRDualRank [FC11]) have adopted the bootstrapping approach above in their sample generation step, because

queries tend to be high-precision (i.e., a high fraction of the retrieved documents is useful for the extraction task at hand). Unfortunately, these techniques compromise recall (i.e., a small fraction of all the tuples in the collection is retrieved) and often miss important relevant groups of useful documents, which is undesirable during the sampling step. We aim to alleviate this issue by exploiting query learning approaches, such as those proposed in QXtract [AG03] and FactCrawl [BLNP11a], and carefully choosing the query execution as well as the document retrieval and processing strategies (Chapter 4).

After document sampling, our extraction process above resorts to query-based approaches for efficient information extraction over each text collection individually (see Figure 2.2). Our second observation is that such a naïve approach would be unnecessarily expensive despite the efficiency-guided efforts made over each collection, because not all collections contain any useful documents. Therefore, to prioritize the extraction effort, for efficiency, we aim to focus on the problem of ranking text collections for an extraction task of interest. An approach for this task should rightfully conclude, for example, that FEMA is better for extracting *Occurs-in* tuples than NYTimes. This collection ranking problem is related to the problem of resource selection in distributed information retrieval [SS11, Chapter 3], to identify topically relevant collections for a given user query. Unlike in distributed information retrieval, though, our information extraction scenario requires that we identify collections with useful documents for the extraction task, rather than documents that are topically relevant for a given query. Despite this difference in focus, we can adapt resource selection approaches to our information extraction scenario, as we will see, as well as develop alternative, information extraction-specific approaches (Chapter 5).

As argued, prioritizing the extraction effort by focusing on the useful collections can potentially reduce the extraction time considerably. Resorting to query-based approaches for efficient information extraction over these (now prioritized) text collections (e.g., QXtract [AG03], PRDualRank [FC11], and FactCrawl [BLNP11a]) would thus be a reasonable next step. Unfortunately, these approaches exhibit critical limitations. First, small document samples, on which these techniques rely, for efficiency, are unlikely to reflect the typically large variations in language and content that useful documents for an extraction task may exhibit. As a result, the queries derived from the document sample may suffer from

low recall during extraction. (We address this problem in part in Chapter 4, as discussed above.) Second, these techniques schedule the documents for extraction once-and-for-all, and thus do not benefit from the information that is captured as the extraction process progresses. Our third observation is that we can exploit this information progressively and in an adaptive manner, to derive other queries that retrieve new useful documents and refine our document schedule so that we process useful documents first (Chapter 6).

Our final observation is related to that in [NVB01], that is, that only small fraction of sentences in a useful document are useful for an extraction task of interest. Based on this observation, we could drastically reduce the extraction time by focusing on these (useful) sentences, and ignoring the rest. In addition to [NVB01], several other efforts (e.g., [XGZ11; WSE13]) have considered this proposition. Unfortunately, these approaches exhibit crucial limitations. The approaches in [NVB01] and [WSE13], for instance, compromise the recall because text filtering techniques are often far from perfect. Similarly, the approach in [XGZ11] compromises the efficiency because it relies on running computationally expensive text processing tasks (e.g., named entity recognition) over the entire text. Even more importantly, these techniques largely ignore already extracted tuples and may lead to extracting the same tuples multiple times, which is many times undesirable. We argue that by examining the document contents at such fine granularity we can effectively prioritize sentences that lead to the extraction of unseen, novel tuples (Chapter 7). We can also extend this idea of novelty to other levels of the data (e.g., documents and collections), as we describe in our future work discussion in Chapter 10.

Distributing Information Extraction: So far we have discussed opportunities to scale information extraction, which promote focusing the extraction effort. Another direction to scale information extraction is that of parallelization and distributed processing, which are in general very attractive choices for processing large volumes of data [Kum02]. Importantly, information extraction is particularly suited for parallelization, since many of the information extraction steps (e.g., part-of-speech tagging, shallow parsing) operate over each document—and sometimes sentences—independently [DEG+03; LZY01; NZRS12a] (see Section 2.1.3). For this reason, most distributed processing architectures (e.g., Hadoop [Whi09], Spark [ZCF+10]) can be adapted for information extraction over large text col-

lections. In our future work discussion (Chapter 10), we review some notable recent efforts in this area (e.g., DeepDive [NZRS12b]) and explore how these distributed approaches can potentially benefit from the targeted strategy that we advocate in this dissertation.

Putting it All Together: Another possible direction, and one that combines all the opportunities described thus far, is to perform a holistic optimization of the extraction process, rather than focusing on each level in isolation. Recent approaches (e.g., [SGG13]) have already taken a step in this direction, specifically, to find—for a given large text collection—the fastest extraction plan (i.e., specific implementations and execution order of operators required for extraction, such as document selection or named entity recognition) that satisfies certain given precision and recall constraints. This is motivated by the fact that the performance of information extraction systems (e.g., the speed or quality of the information extraction system)—and in turn of the overall extraction process—tends to vary considerably across text collections [IAJG07; AC05; JS09]. For instance, while a collection C_1 may include substantially more useful documents than another collection C_2, and should hence be deemed as more useful during collection ranking, the extraction process over C_1 may fail to find many of its useful documents and, as a result, produce fewer tuples than over C_2. Prioritizing the collections and documents to process with an information extraction system of choice could hence be done comprehensively, integrating signals from all data levels. Simultaneously, and if the extraction process were run over a distributed infrastructure, many more options should be considered. For example, the cost of fetching a set of documents or sentences would now have to be considered together with their usefulness, to decide whether it is cost-effective to fetch additional text contents or process local contents instead. Indeed, many other interesting challenges arise as a result of this holistic proposition, and we plan to explore them further in future work (Chapter 10).

2.4 Applications

A large variety of applications across a wide range of areas are empowered by information extraction, specifically, by running information extraction systems over large volumes of text data, such as the text contents on the Web. In this section, we review some of the

most relevant such applications.

Knowledge Base Population: *Knowledge bases* are information repositories that store facts about the world (e.g., entities and their attributes, relations between entities) in machine-readable formats. Some examples of knowledge bases are: (i) academic efforts, like KnowItAll [ECD+05; EBSW08; FSE11], ConceptNet (MIT) [SH12], DBpedia [ABK+07], Freebase [BEP+08], NELL [CBK+10], WikiTaxonomy [PS07], YAGO [SKW07], YAGO2 [HSBW13], and DeepDive [NZRS12b]; as well as (ii) commercial projects, such as those by Microsoft [Qia13], Google [Sin12], Facebook [Dar13], and Walmart [DLT+13]. Knowledge bases are an important building block of many applications (some of which we describe in detail below): Machine translation (e.g., [CGN05]) and word sense disambiguation (e.g., [BP06]) use lexical knowledge, query expansion exploits various taxonomic knowledge (e.g., [LLYM04; GSW05; TSW06]), document classification based on supervised or semi-supervised learning can be combined with domain-specific ontologies (e.g., [IW06]), and question answering and information retrieval combine various sources of structured knowledge (e.g., [HLN; ALG01; BDB02; MNPT02]).

The increasing interest in knowledge bases in the past few years has demanded automated efforts for building and enriching these knowledge bases, a task often referred to as *knowledge base population*. One family of approaches to knowledge bases population focuses on extracting information from large volumes of unstructured text (e.g., the Web) for a given ontology or schema representation using information extraction systems.[2] Some of the most relevant approaches in this family are NELL [CBK+10], PROSPERA [NTW11], DeepDive [NZRS12b], and Knowledge Vault [DGH+14]. Due to the massive amounts of text contents that approaches in this family need to process, efficiently deploying information extraction systems is thus of critical importance.

Question Answering: Question answering systems aim at producing precise answers to questions often posed in natural language [KM11]. One type of such questions is the so-called *list questions*, which expect a list as an answer. For example, for a list question *[Name*

[2]Other approaches to knowledge base population build knowledge bases from already structured sources (e.g., YAGO [SKW07], YAGO2 [HSBW13], DBpedia [ABK+07], and Freebase [BEP+08]), from open-ended structures (e.g., Reverb from KnowItAll [FSE11], OLLIE [MSB+12], and PRISMATIC [FFGK10]), and from manual annotations (e.g., Freebase [BEP+08], Cyc [HZW10], WordNet [MBF+90])

the coffee-producing countries in South America], a list question answering system (e.g., [WSCN08; YC04]) should return *"Brazil, Colombia, Peru, and Venezuela,"* among others. Another type of questions is the so-called *factoid questions*, whose answers are entities (e.g., person, location, organization) or attributes thereof (e.g., place of birth, population, net worth). For instance, for a question *[Who won the 1980 Nobel Peace Prize?]*, a factoid question answering system (e.g., [CA05; IBGC+14]) should return *"Adolfo Pérez Ezquivel."* Due to their structured nature, answers for both types of questions often derive from a knowledge base, which is many times populated via information extraction, as discussed. Other times, specifically, in real-time question answering [ESI+12], answers need to be produced in seconds, by processing large text collections. For both scenarios, the efficiency of running information extraction systems over text corpora is crucial.

Decision Making: Decision making is the process of selecting a logical choice from a set of available options. In the financial domain, in particular, decision makers need an intuition on the state of the financial market, which is many times sensitive to breaking news on economic events (e.g., company acquisitions, stock splits, dividend announcements). Because of the large volumes of text information that have to be processed, automating the detection of these events with information extraction is crucial to enable faster processing and making better informed decisions. Not surprisingly, many research and commercial efforts (e.g., [SFMB07]) have addressed this critical proposition [Cvi10]. As text contents continue to grow at high speeds, and as different events are regarded as valuable signals for decision making, research has to focus on the efficiency of the deployment of the extraction process, to satisfy the time-critical necessities of the financial market.

Scientific Applications: Besides the commercial applications above, there are also scientific applications for which information extraction is a critical building block. Two such applications are *bioinformatics* and *scientometrics*. Bioinformatics is the field that studies the processing, understanding, and organization of information in biomedical literature via computational algorithms, to facilitate the access and exploration of the vast biomedical literature [LGG+01; HDG00]. A central problem in bioinformatics is the extraction of entities, such as genes and proteins, as well as their interactions, which are well-known applications of information extraction. However, many other scientific tasks respond to templates that

can be successfully addressed by information extraction: For example, scientists working on drug discovery have an ongoing interest in reactions catalysed by enzymes in metabolic pathways. These reactions may be viewed as relations in which various entity types (e.g., enyzmes, compounds) play particular roles (e.g., substrate, catalyst, product). Importantly, bioinformatics is a very active research area that has helped advanced the state-of-the-art of information extraction algorithms vigorously.

The other notable scientific application, namely, scientometrics, is the field that studies the measurement and analysis of science (e.g., to predict and assess impact of authors, articles, journals, or fields). Early approaches (e.g., [DYFC09; CWW02]) have relied mainly on the metadata of the scientific articles (e.g., bibliography, keywords, institutions, journal characteristics) and other much richer features derived from them (e.g., collaboration networks [VHL10]). More recent approaches (e.g., [YHO+11] and [MIC+15]) have moved a step forward, to focus also on features derived from the text of the articles. In particular, and for the task of author and term prominence detection, [MIC+15] uses information extraction over large volumes of scientific articles to extract entities (e.g., algorithms, datasets, genes) and relations (e.g., novelty claims, dataset purposes) as features for the pertinent prominence analysis. Such features can indicate, for instance, the number of algorithms that have been implemented for a given input problem, which may be a meaningful indicator of the depth in which the problem has been studied. Importantly, [MIC+15] has shown these (extraction-based) indicators to be among the top-performing ones for this task. Therefore, there is crucial interest in effectively devising and efficiently deploying extraction tasks for computing this type of indicators.

Reading Experience Enhancement: The last family of applications promotes using structured data to help humans better understand the unstructured text. Two such applications, namely, *semantic culturomics*[3] and *reading comprehension improvement*, have recently gained substantial attention, due to the copious statements about a large number of entities and relations that knowledge bases have accumulated. Semantic culturomics, on the one hand, is defined in [SP14] as "the large-scale analysis of text documents with the help of knowledge bases, with the goal of discovering, explaining, and predicting the trends

[3]Semantic culturomics is also referred to as *knowledge-based culturomics* [TBC+15].

and events in history and society." Semantic culturomics could, for example, answer questions such as *"which are the less-densely populated cities near geographical areas affected by natural disasters?"* by first detecting *Occurs-in* tuples and then retrieving—from the knowledge base—the population density of cities near those mentioned in the tuples. Importantly, this dynamic interaction with text will produce more—and more diverse—user needs, where information extraction will play a significant role both (i) populating knowledge bases, to improve coverage; and (ii) finding entities and relations in text, to enable semantically richer questions.

The second application, namely, reading comprehension improvement, consists of identifying concepts in text (e.g., measurements) that are difficult to understand for traditional readers and producing a more meaningful definition of the concept. For example, a system for this application would perceive that *"300 million firearms"* in text excerpt *"Americans own almost 300 million firearms"* is difficult to comprehend and, in turn, would put it into perspective, in a clearer way, as: *"300 million firearms is about one firearm for every person in the U.S."* Some of the existing efforts to address this task (e.g., [LZBB13; Gen15]) rely on crowdsourcing to both identify and explain concepts. As shown in this example, though, both the concepts and its explanation are structured: The use of information extraction to identify them is therefore imminent. Recent approaches (e.g., [HDA13; KKJ+15]) have moved towards automatic approaches empowered by information extraction, to exploit—in a scalable manner—the vast amounts of text generated on a daily basis. Specifically, these approaches employ information extraction for: (i) automatically identifying concepts that are, in turn, characterized based on their complexity; and (ii) populating knowledge bases, as argued above, with valuable facts for this task.

As discussed, many interesting applications rely on running information extraction systems over large volumes of text data. These applications would directly benefit from efficient and scalable approaches that deploy these (often expensive) information extraction systems over the large text collections, the focus of this dissertation.

Chapter 3

REEL: A Toolkit for Developing Relation Extraction Systems

As discussed in Section 2.1.1, relation extraction is a complex task that relies on a wide variety of text processing and machine learning steps to extract tuples from text. This complicates the definition, deployment, and evaluation of systems for new extraction tasks and text collections, crucial for a large variety of text-centric tasks, as argued. The problem in which we focus in this chapter is, therefore, that of facilitating the definition, deployment, and evaluation of relation extraction systems.

Specifically, the contribution of this chapter is REEL (**RE**lation **E**xtraction **L**earning framework), an open-source framework to easily develop and evaluate relation extraction systems. REEL provides the code and infrastructure to: (i) handle various input text formats, which enables operations over different text collections; (ii) plug in appropriate text processing steps and tools, which enables diverse processing of the text with minimal effort; (iii) define and combine conceptual relation constraints that are automatically enforced; (iv) decouple learning and extraction from the text processing, which enables the straightforward integration and re-usability of different extraction algorithms; and (v) uniformly execute and evaluate relation extraction systems, which enables the testing and fair assessment of these systems. REEL, in contrast to existing toolkits, effectively modularizes the key components involved in relation extraction systems. To define a relation extraction

system for a new relation and text collections, users only need to specify the parsers to load the collections, the relation and its constraints, and the learning and extraction techniques, which makes REEL a powerful framework to enable the deployment of relation extraction systems for both research and application building.

The rest of this chapter is organized as follows. Section 3.1 characterizes the limitations of existing toolkits often used to develop relation extraction systems and defines the problem that we study in this chapter. Section 3.2 provides a high-level description of REEL's architecture. Sections 3.3 and 3.4 describe the text processing and learning and extraction components, and how they integrate to enable relation extraction. Finally, Section 3.5 shows how to use REEL in practice, by providing an end-to-end implementation of a typical relation extraction system, and Section 3.6 concludes the chapter. REEL is publicly available as open source under the General Public License Version 3 (GPLv3) license, at `http://reel.cs.columbia.edu/`. The bulk of this chapter has been published as [BSGG14].

3.1 Background and Problem Definition

Many relation extraction systems have been proposed in the literature (e.g., [BM05b; FSE11; NWS12; ZAR+03; ZG05]). However, few such systems are publicly available and, even when they are, it is usually problematic to adapt and evaluate them over new relations and text collections. Because of all the steps involved in relation extraction, and because of the wide variety of tuple extraction algorithms, developing relation extraction systems is a rather challenging and time-consuming process (see Section 2.1). To avoid implementing such complex systems from scratch, and to facilitate their adaptation and evaluation, developers often rely on relation extraction toolkits. Many such toolkits (e.g., [LKT+15; BRMB11; Sod99; SGD11; LCY+12; YPWC+13; AKM13; AMB14; GM14]) target novice developers by providing interactive interfaces to develop extraction systems with minimal coding effort. These systems, in particular, mainly focus on facilitating the compilation of effective extraction patterns. Other toolkits are more flexible, and provide code and infrastructure to develop relation extraction systems as extensions of the original code. One such

toolkit is T-Rex [Iri05], which is based on the text processing and entity extraction learning framework RUNES [IC06]. T-Rex models the various relation extraction steps (e.g., word tokenization, entity recognition, learning of the relation extraction model) as additional text processing steps that can be coupled together—via a common interface—with RUNES's native components.

Unfortunately, T-Rex splits the relation extraction task into relatively coarse modules, which limits the reuse of text processing and learning components across relation extraction tasks, and hence complicates the implementation of new extraction tasks. In particular, since all components share a generic interface that does not reveal their internal functionality properly, the developer needs to be aware of the low-level details of these components to reuse them successfully. Also, the coarse T-Rex modules do not constrain their output, which complicates the experimental comparison of different relation extraction strategies. As a result, to experimentally evaluate and compare relation extraction systems in T-Rex, we must rely on ad-hoc solutions, which is undesirable. Furthermore, changes in the evaluation process, such as using new evaluation measures, may lead to ubiquitous and fine-grained source code modifications across systems, which is problematic.

Other toolkits originally proposed for related text-centric tasks provide low-level building blocks that are helpful for relation extraction, but lack the code and infrastructure to support all steps involved in relation extraction. Specifically, text processing toolkits (e.g., UIMA [FL04]) tend to only provide support for the entity tagging and feature extraction steps described above and, as a result, lack infrastructure for the remaining steps. Machine learning libraries (e.g., MALLET [McC02], LibSVM [CL11], Weka [HFH+09]), in contrast, provide the foundation for learning the tuple extraction algorithm, although they lack support for the variety of steps that relation extraction systems routinely need, such as entity tagging and candidate enumeration. Finally, natural language processing suites (e.g., NLTK [Bir06], OpenNLP [ope15b], StanfordNLP [sta15a], ClearTK [OWB09], LingPipe [lin15]) and entity extraction frameworks (e.g., RUNES [IC06], MinorThird [Coh04]) consolidate the features of the text processing and machine learning libraries, although they lack support for relation extraction altogether, in that they do not offer infrastructure for, say, candidate generation and relation constraints.

To support relation extraction, one alternative would be to extend the toolkits above by including the missing components. However, this would require in many cases a significant implementation effort and a drastic redesign of the toolkit, since we would have to incorporate full support for the missing steps. A more promising approach, which we advocate in this dissertation, is to integrate and complement valuable text processing toolkits—to exploit their powerful implementations of low-level text operations—and machine learning toolkits—to exploit their powerful implementations of relevant learning operations—for our relation extraction task. Our approach is then similar in spirit to that of ClearTK [OWB09] but for a different problem. (ClearTK focuses on statistical natural language processing.)

To effectively identify the challenges of developing a toolkit for this difficult task—and to explain in practical terms how our proposed solution addresses these challenges—we rely on the following running example:

Example 2 *Consider the relation Charged(Person, Charge, Date) of Section 2.1, which, as we discussed, contains a tuple ⟨p, c, d⟩ if person p was accused of charge c on date d. A properly trained relation extraction system for such relation should extract the tuple ⟨Mark Chapman, second-degree murder, 1981⟩ from the text excerpt "John Lennon's killer, Mark Chapman, was sentenced in 1981 to 20 years to life in prison after pleading guilty to second-degree murder."*

The next sections explain the key steps to tackle our problem of focus, and provide a roadmap to effectively address different practical challenges of the relation extraction problem.

3.2 System Architecture

We now introduce REEL's flexible, modular architecture (see Figure 3.1). REEL's components can be divided into two types. The Text Processing components are responsible for transforming the input text documents into the input format for the relation extraction techniques. Then, the Learning and Extraction components perform the (arguably) most important relation extraction tasks, namely, to learn, execute, and evaluate relation extraction systems. We now provide details about these two types of components.

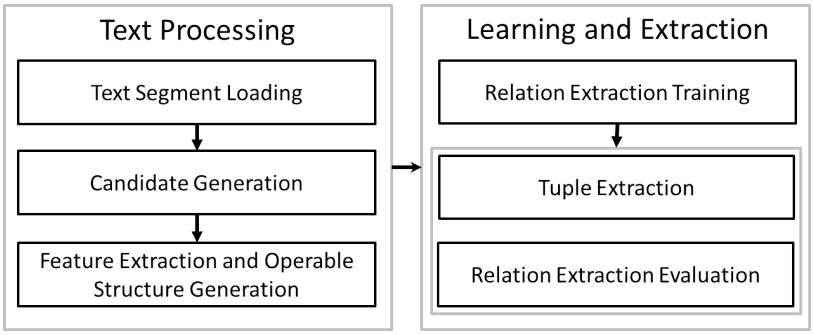


Figure 3.1: REEL system architecture.

The Text Processing components (see Figure 3.1), which include the Text Segment Loading, Candidate Generation, and Feature Extraction and Operable Structure Generation components, focus on processing the input documents for the relation extraction system. First, the Text Segment Loading component, which we describe in Section 3.3.1, loads the documents in a text collection and transforms them into text segments that the relation extraction techniques can then process. The key challenge here is to allow the integration of different text processing tools (e.g., file loaders, XML parsers) so that it is easy to use different types of collections in REEL. Second, the Candidate Generation component, which we describe in Section 3.3.2, produces the candidate text segments that we introduced in Section 2.1.2. As discussed, these candidate text segments satisfy the constraints associated with the relation at hand (e.g., that the text segment must include the *Person*, *Charge*, and *Date* entities within N words of each other). The key challenge for this component is to enable a flexible definition of constraints over the entities and the relation itself. Finally, the Feature Extraction and Operable Structure Generation component, which we describe in Section 3.3.3, extracts the features required by a specific relation extraction algorithm and produces the data structure for the extraction algorithm (e.g., sequences or trees of features). These data structures, which we refer to as *operable structures*, are a feature-enriched version of the candidate text segments on which the learning and extraction algorithms will operate. The key challenge is then to provide a unified interface for the extraction of features that supports the wide variety of features and structures that different learning algorithms may require.

The Learning and Extraction components (see Figure 3.1), which include the Relation

Extraction Training, Tuple Extraction, and Relation Extraction Evaluation components, focus on the relation extraction algorithms. First, the Relation Extraction Training component, which we describe in Section 3.4.1, automatically produces a *relation extraction model* using, as training input, labeled operable structures, which indicate whether the relation of interest holds among their entities. Second, the Tuple Extraction component, which we describe in Section 3.4.2, uses the model produced in the Relation Extraction Training component to extract tuples corresponding to related entities. Notably, Tuple Extraction performs a classification task over unlabeled operable structures and produces tuples of entities that are likely related. Third, the Relation Extraction Evaluation component, which we describe in Section 3.4.3, evaluates the relation extraction systems according to a given set of evaluation metrics. When proposing an architecture for the learning and extraction components, the key challenge is to provide a unified interface for different relation extraction techniques to help train, execute, and evaluate the resulting models with minor changes in the code.

Next, we describe the different components discussed thus far in detail. Later, in Section 3.5, we show the implementation of the *Charged* running example in REEL, which demonstrates how our framework helps in writing relation extraction systems in a simple and easy-to-understand manner.

3.3 The Text Processing Component

We now describe the Text Processing components, which focus on processing the input documents for the relation extraction system.

3.3.1 Text Segment Loading

The first component of REEL, Text Segment Loading, is the interface between the original representation of documents (e.g., XML files, file directories, or document indexes) and the internal representation that REEL employs to represent the text segments. The main goal of this component is to detach the formatting subtleties of the text collections from the further operations to be run over them by producing text segments that include mentions

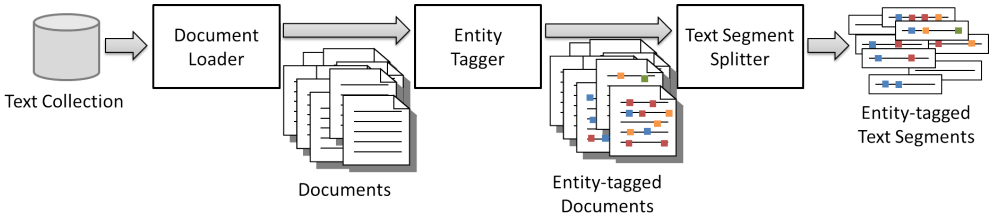


Figure 3.2: Text Segment Loading.

to the entities of interest.

The Text Segment Loading component in REEL starts by reading the text documents from their original source (see *Document Loader* in Figure 3.2). Once the documents are loaded, REEL performs the tagging of entities required by the relation of interest (see *Entity Tagger* in Figure 3.2). For example, in our running example, the user needs to provide entity taggers to annotate *"John Lennon"* and *"Mark Chapman"* as *Person*, *"second-degree murder"* as *Charge*, and *"1981"* as *Date* entities. This tagging process can differ substantially according to the text collections (e.g., some datasets such as ACE 2005 [Wal06] already contain named entity annotations) and available tagging resources (e.g., some toolkits provide off-the-shelf, pre-trained named entity recognition models). REEL supports different types of entity taggers that range from simple loaders from the document files to models from open-source named entity tagging frameworks, such as Stanford NER [Sta15b] and E-txt2DB [Etx12].

Finally, REEL splits the output of the text document loader into text segments according to the needs of the extraction (see *Text Segment Splitter* in Figure 3.2). These text segments can be different subsets of the documents, such as sentences or paragraphs, and can vary according to the extraction task. This task may also depend on the text, and REEL accommodates these different scenarios. For instance, some collections such as Aimed [Moo11] provide the input documents already split into sentences.

The text segments resulting from the Text Segment Loading component contain all the entities that can potentially appear in an extracted tuple. In the next section, we discuss how to generate candidate text segments (i.e., text segments that represent a potential relation between a specific set of entities).

3.3.2 Candidate Generation

Now that we have explained how the Text Segment Loading component produces text segments annotated with entities, we describe the generation of candidate text segments. This component receives as input the tagged text segments and the constraints of the relation. Then, for each input text segment, it produces all the candidate text segments that comply with the input constraints (Figure 3.3). REEL supports two types of constraints, namely, *Entity Constraints* and *Relation Constraints*, that help to conceptually define the relation of interest. REEL offers users the flexibility to define their own constraints and combine them with others of the same type via logical Boolean expressions. We define these constraints as follows:

- **Entity Constraint:** Entity Constraints are the conditions that entities need to satisfy to be part of the relation of interest. Examples of such constraints are the entity type constraints, which define to which types an entity can belong (e.g., all *Charged* relations must be between a *Person*, a *Charge*, and a *Date*); and non-mandatory constraints, which define whether the occurrence of an entity is optional (e.g., in the *Charged* relation, we may omit the *Date* but neither the *Person* nor the *Charge* can be omitted).

- **Relation Constraint:** Relation Constraints are the conditions that apply to the relation as a whole. Among these constraints we may find distance constraints (e.g., the distance between entities should not exceed 10 words) and number of occurring entities (e.g., at least two entities need to participate in the relation even if some are optional).

Candidate text segments represent potential tuples for the relation at hand in the text segment, and, in effect, a single text segment may derive multiple candidates. REEL automatically computes these candidate text segments (see Algorithm 1), thus effectively hiding the complexity of the process when multiple entities and constraints are involved. Algorithm 1 receives as input a text segment with annotated entities and the relations of interest. For example, a valid input would be *"John Lennon's killer, Mark Chapman, was sentenced in 1981 to 20 years to life in prison after pleading guilty to second-degree murder"*

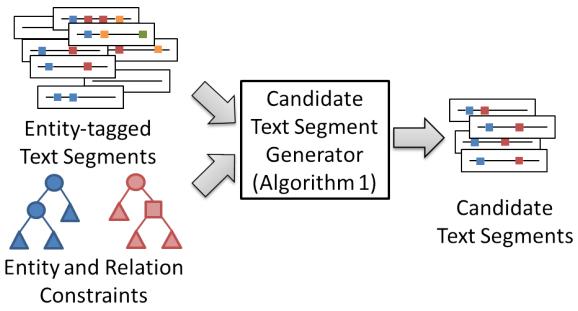


Figure 3.3: Candidate Generation.

and the *Charged* relation. Then, for each of the relations of interest (line 2), the algorithm obtains all possible entities in the text segment that comply with their corresponding entity constraints (lines 5-7). In our example, we would obtain *"John Lennon"* and *"Mark Chapman"* for *Person*, *"second-degree murder"* for *Charge*, and *"1981"* for *Date*. Then, the algorithm produces all possible combinations (see Mappings function in line 10) according to the relation constraints (line 9). For example, if the relation constraints indicate that entities should occur within 10 words and that *Date* is optional, the possible mappings will be: ⟨John Lennon, second-degree murder, 1981⟩, ⟨Mark Chapman, second-degree murder, 1981⟩, ⟨John Lennon, second-degree murder⟩, and ⟨Mark Chapman, second-degree murder⟩. Then, the algorithm creates the candidate sentence and incorporates the corresponding entities (lines 11-13). Finally, the candidate sentences are added to the result set (line 14). Notice that some candidate text segments are subsumed by others in that their tuples are a subset of the tuples in other candidate text segments. For example, the candidate text segment that includes the potential tuple ⟨Mark Chapman, second-degree murder⟩ is subsumed by another candidate text segment that includes the potential tuple ⟨Mark Chapman, second-degree murder, 1981⟩. Interestingly, the entities in the tuple may only be related in the absence of the *Date* entity, and thus reporting only the larger tuple may miss crucial information. Thus, the decision on whether to report these two tuples as different tuples or only return the tuple that includes all entities is left to the relation extraction system implementation per se, as we describe in Section 3.4.2.

The output of the Candidate Generation component includes sets of entities that may constitute a relation. However, as discussed, the relation extraction algorithms usually need

Algorithm 1: genenerateCandidates(*Seg*, *Rels*)

```
1:  Cands← ∅
2:  for each relation Rel ∈Relations do
3:    CandE← ∅
4:    for each role r∈Rel.roles do
5:      C←Rel.entityConstraint(r)
6:      Ents←C.compatibleEntities(Seg.Entities)
7:      CandE← Cand ∪ ⟨r, Ents⟩
8:    end for
9:    RelC← Rel.relationConstraints()
10:   for each mapping m∈Mappings(CandE, RelC) do
11:     for each entity E∈m do
12:       N←Rel⟨E.role, E.entity⟩
13:     end for
14:     Cands←Cands∪N
15:   end for
16: end for
17: return Cands
```

additional hints, in the form of features, to decide if there is a relation between the entities of a candidate text segment. In the next section, we describe how we enrich candidate text segments with such features.

3.3.3 Feature Extraction and Operable Structure Generation

So far, we have described how REEL generates the candidate text segments for a relation of interest. These candidate text segments only include the entities that satisfy the entities and relation constraints and hence, are not specific to any relation extraction technique. To make the candidate segments usable for a relation extraction technique, REEL needs to "enrich" these candidates with features (e.g., lemmas, part-of-speech tags, dependency graphs) and produce their corresponding operable structures (Figure 3.4).

For this, REEL considers the requirements of the relation extraction technique of interest, including its features and how to store them in its operable structure. Such information is carried in what we refer to as the *core* of the relation extraction technique. In particular, cores are responsible for two crucial tasks. First, they guarantee that operable structures include the mandatory features. For example, if our relation extraction system requires

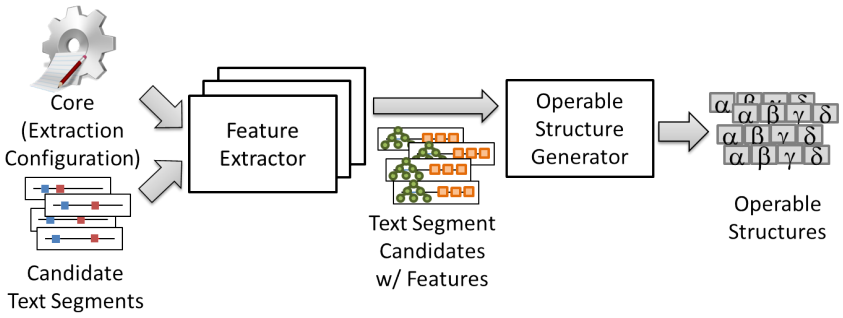


Figure 3.4: Operable Structure Generation.

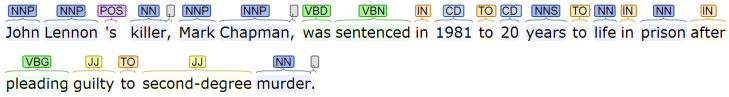
tokens and their part-of-speech tags, the core must report these features as mandatory. Second, cores guarantee that the operable structures are represented in appropriate data structures for training. For example, if the training algorithm requires numeric vectors to represent each training instance, the core must store the operable structures in that form. To define cores along with their mandatory features, REEL provides a simple interface that is shared across relation extraction systems. This interface is general enough to enable the incorporation of additional features to existing cores, which in turn helps to effortlessly experiment with these features in other extraction systems.

Most of the features for relation extraction are typically shared across techniques, as discussed in Section 2.1.2, and as such, should be computed uniformly. For this, REEL provides *Feature Extractors* (Figure 3.4). Feature extractors respond to a unified interface and can be re-used for different cores with no additional effort. Each core is then responsible for storing the extracted features in their own format, as discussed. REEL defines, but is not limited to, three types of features, namely, *vector-based*, *sequence-based*, and *graph-based*, which we define as follows:

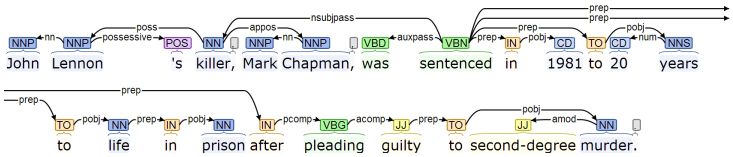
- Vector-based features refer to the most common feature representation in classification tasks. In this representation each characteristic of the candidate text segment is represented as a number (usually in a binary representation) and the entire set of m features forms an m-dimensional space. Several external tools use this representation, as is the case for the machine learning toolkit Weka [HFH+09] and its Instance object. For example, we can create in REEL an Instance-based operable structure as

```
@attribute useful {YES,NO}
@attribute john {0,1}
@attribute lennon {0,1}
@attribute killer {0,1}
@attribute mark {0,1}
@attribute chapman {0,1}
@attribute sentenced {0,1}
@attribute 1981 {0,1}
@attribute 20 {0,1}
@attribute years {0,1}
@attribute life {0,1}
@attribute prison {0,1}
@attribute pleading {0,1}
@attribute guilty {0,1}
@attribute second-degree {0,1}
@attribute murder {0,1}
@data
0 YES, 1 1, 2 1, 3 1, 4 1, 5 1, 6 1, 7 1, 8 1,
    9 1, 10 1, 11 1, 12 1, 13 1, 14 1, 15 1
```

(a) Weka instance as example of vector-based features.



(b) Part-of-speech tags as example of sequence-based features.



(c) Dependency graph as example of graph-based features.

Figure 3.5: Examples of features in REEL.

illustrated in Figure 3.5a, where each `@attribute` line corresponds to a feature and the last line corresponds to pairs that include the index of the feature and a Boolean value that indicates whether the feature occurs in the text or not.

- Sequence-based features refer to the text segment features that are modeled as sequences. As an example, consider part-of-speech tags, which produce a sequence of features, one for each token in the sentence. Figure 3.5b illustrates the part-of-speech tags of the text excerpt *"Mark Chapman was sentenced in 1981 to 20 years to life in prison after pleading guilty to second-degree murder,"* where each part-of-speech tag corresponds to one term or punctuation symbol in the text excerpt.

- Graph-based features refer to the text segment features that are modeled as a graph. As an example, consider dependency graphs, which move away from the linearity of sequence-based features to a more complex feature space. Figure 3.5c illustrates the dependency graph of the text excerpt *"Mark Chapman was sentenced in 1981 to 20 years to life in prison after pleading guilty to second-degree murder."* As we see in this figure, there are (directed) connections between part-of-speech tags that together form a graph of features.

Interestingly, dependencies between features are common in information extraction. For example, the dependency graph needs the part-of-speech tags to be computed first (see Figure 3.5c). To avoid computing the same features multiple times, which may incur a

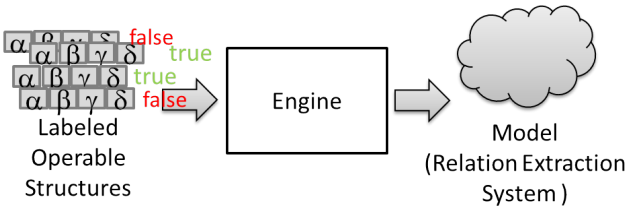


Figure 3.6: Training a relation extraction system.

substantial overhead, REEL provides feature caching, so that each feature set is computed only once.[1] Specifically, REEL maintains two caches: one cache stores the features extracted from the candidate text segment, which depend on the tagged entities (e.g., distance between entities), whereas the other cache stores the features derived from the text segment, which do not depend on the entities (e.g., tokens or part-of-speech tags). Such a distinction is necessary, since the cache for text segment features is stored only once and shared across its (derived) candidates.

After Feature Extraction and Operable Structure Generation, the documents are fully processed and can serve as input to relation extraction algorithms. In the next section, we explain how to train relation extraction systems with REEL given the results of the Feature Extraction and Operable Structure Generation component.

3.4 The Learning and Extraction Component

We now describe the Learning and Extraction components, which focus on the relation extraction algorithms, as discussed.

3.4.1 Relation Extraction Training

As we argued in Section 2.1.1, there are multiple techniques to train relation extraction systems and our framework should be flexible enough to support different learning settings (e.g., techniques, training algorithms, existing libraries). REEL includes two concepts, namely, *engine* and *model*, that together with the core (Section 3.3.3) provide these capa-

[1]Such caching is possible because feature extractors are deterministic (i.e., they produce the same output given the same text segment).

bilities. In a nutshell, the engine runs the training algorithm: Given a set of labeled operable structures—generated under the constraints of a core—the engine produces a relation extraction model (Figure 3.6). The engine must then be aware of the internal representation of the operable structure to guarantee the effective usage of the included features. For example, if the operable structure is represented with graph-based features (Section 3.3.3), the learning algorithm in the engine should be aware of such structure to manage it successfully.

Engines bring flexibility into the development of relation extraction systems along several dimensions. First, engines allow developers to use their machine learning libraries of choice (e.g., JlibSVM [Soe14], Weka) with no restrictions, or to develop their own techniques to learn models (e.g., pattern-based techniques such as PRDualRank [FC11]).[2] Second, engines enable the modification of several learning decisions for a given core with minimal effort. For example, for an SVM-based core, we can customize existing engines with different learning strategies (e.g., batch vs. online) and different learning parameters (e.g., convergence criterion).

The flexibility of the engines described above is carried over to the learned models, which include the necessary information to perform the classification task. For example, if the relation extraction system relies on an SVM-based classifier [Joa98b], the model will include the support vectors, whereas if the relation extraction system relies on patterns, the model will include the learned patterns and how they are matched with the text.

The next section discusses how to use the relation extraction model that results from the Relation Extraction Training to extract tuples from text.

3.4.2 Tuple Extraction

As we discussed in Section 3.2, REEL performs the tuple extraction as a classification task: REEL uses the model learned by the Relation Extraction Training component (Section 3.4.1) to classify unlabeled operable structures as positive (i.e., their entities are related) or negative (i.e., their entities are not related). Therefore, during tuple extraction, REEL observes a set of unlabeled operable structures and whenever one of these structures is classified as containing related entities, REEL produces a tuple with those entities (see

[2]Our distribution of REEL includes an end-to-end implementation of PRDualRank for reference.

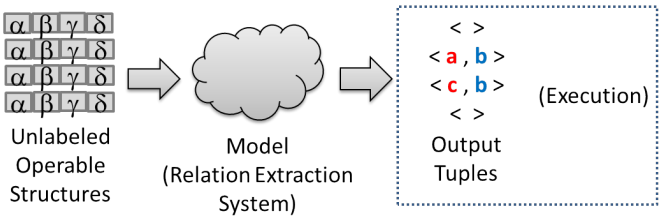


Figure 3.7: Tuple Extraction.

Figure 3.7).

For example, consider a relation extraction model that receives operable structures based on dependency graphs and decides if their entities are an instance of the *Charged* relation in our running example. For our example sentence from Section 3.1, the operable structures would resemble that in Figure 3.5c. Since there are four candidate text segments for this sentence (see Section 3.3.2), the model would evaluate all the alternatives and output as relation instances ⟨Mark Chapman, second-degree murder⟩ and ⟨Mark Chapman, second-degree murder, 1981⟩. Furthermore, the model would attach information on tuple subsumption to the tuples, so that applications or end users can decide what tuples to consider.

Beyond the extraction of individual relations, we now discuss another important feature of REEL, namely, how it supports the comprehensive experimental evaluation of alternative relation extraction models.

3.4.3 Relation Extraction Evaluation

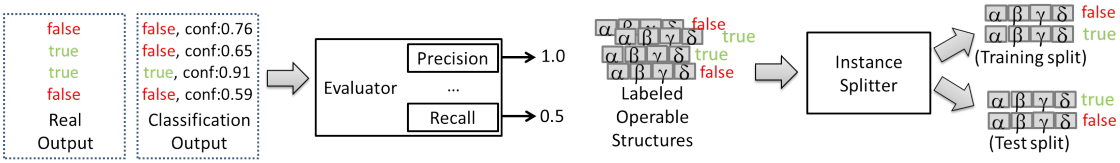
In addition to the definition of individual relation extraction systems, REEL supports the experimental evaluation and comparison of multiple relation extraction systems, a task of critical importance to facilitate research in information extraction. For this, REEL provides the notion of *Evaluators*. An evaluator helps to: (i) compare different configurations of the same relation extraction system to find the best performing setup; (ii) compare the performance of a relation extraction system over different text collections to demonstrate robustness; and (iii) compare different relation extraction systems over the same text collection to identify the best performers. To achieve this, the evaluator receives as input

both the real and predicted labels of a set of operable structures together with their prediction properties (e.g., confidence of the output), and returns the measured performance (see Figure 3.8a).

The REEL Relation Extraction Evaluation component considers two important factors for the evaluation of relation extraction systems, namely, how to split sets of instances (e.g., documents, candidate sentences, or operable structures) and what performance measures to use. Regarding the first factor, REEL provides the generation of instance splits to facilitate the generation of principled training and test sets, as illustrated in Figure 3.8b. Specifically, REEL is able to create these splits over sets of documents, candidate text segments, and operable structures, thus offering different evaluation capabilities. For example, splitting over candidate text segments allows segments from the same text document to belong to both the training and test sets, which would not be possible if we could only split over entire documents. REEL currently provides two types of splitting, namely, *percentage split*, which splits elements at a given fraction (e.g., 70% for training and the rest for testing), and *K-fold split*, which splits elements into K equally-sized groups suitable for K-fold cross validation [Sto74].

Regarding the choice of performance measures, REEL provides common measures for binary classifiers, namely, precision, recall, and F-measure, which can be used in the evaluator, as illustrated in Figure 3.8a. In addition to these metrics, REEL allows the computation of micro and macro averages over them, which is crucial during the evaluation of relation extraction systems that extract multiple relations simultaneously. Altogether, these performance measures enable the principled evaluation of relation extraction systems. REEL also provides support for additional performance measures. Specifically, users can implement measures that take as an input a set of operable structures along with their true labels, predicted labels, and other prediction properties (e.g., confidence of the prediction, probability of entities being related), and ultimately produce an output value. Moreover, users can define new prediction properties that the models can then explicitly report. These new prediction properties would incur source code changes; however, these changes would only affect the models, since that is where the actual prediction takes place.

Now that we have discussed the main components of REEL, we show how we use REEL



(a) Evaluator and Performance Measures. (b) Instance Splitter.

Figure 3.8: Evaluation capabilities in REEL.

to develop a relation extraction system for our running example.

3.5 Using REEL in Practice

In the previous section, we introduced the architecture of REEL and described its components in detail. We now illustrate how to use REEL in practice by providing an end-to-end implementation of a typical relation extraction system for our *Charged* relation example. Specifically, we show the Java source code to perform text segment loading and candidate generation, feature extraction and operable structure generation, relation extraction training, tuple extraction, and relation extraction evaluation.

Candidate Generation: To address these tasks a REEL user should use code templates provided in the toolkit. Then, the users could adapt these templates to their own needs by using different implementations of document loaders, constraints, feature generators, engines, and cores. Users can also implement their own version of these elements to exploit new techniques. First, as explained in Sections 3.3.1 and 3.3.2, we load the text segments of an input collection and derive their candidates (Sample Code 3.1). We define the *Charged* relation along with its constraints, which we save via serialization for future use (lines 2-9). We then load the documents from our collection (lines 11-13), each with their corresponding text segments. Users can write their own collection loaders, which are only required to produce documents in the REEL format as output. Next, we create a text splitter (lines 15-16), which defines the scope of the candidate text segments (e.g., sentences) and, in turn, where the (potential) tuples can occur. Finally, we use the REEL candidate generator to produce the candidates from all documents in the collection (lines 17-21) and save them for the following steps (line 22).

Sample Code 3.1: Candidate Generation.

```
1  // Define relationships and their constraints
2  String r = "CHARGED", t1 = "PER", t2 = "CHAR", t3 = "DAT";
3  RelationshipType rT = new RelationshipType(r, t1, t2, t3);
4  rT.setConstraints(new EntityTypeConstraint("PER"), t1);
5  rT.setConstraints(new EntityTypeConstraint("CHAR"), t2);
6  rT.setConstraints(new EntityTypeConstraint("DAT"), t3);
7  Set<RelationshipType> rTs = new HashSet<RelationshipType>();
8  rTs.add(rT);
9  SerializationHelper.write("rTypes.ser", rTs);
10 // Use a predefined Document Loader
11 Loader l = new MyLoader(rTs);
12 File AD = new File("/train/");
13 Dataset<Document> col = new Dataset<Document>(l, AD, false);
14 // Define sentence splitter for the candidate generator
15 String sp = "model.bin"
16 OpenNLPMESplitter spl = new OpenNLPMESplitter(sp);
17 CandidatesGenerator g = new CandidatesGenerator(spl);
18 Set<CandidateSentence> cand = new HashSet<CandidateSentence>();
19 for (Document d : col) {
20   candidates.addAll(g.generateCandidates(d, rTs));
21 }
22 SerializationHelper.write("train.ser", cand);
```

Operable Structure Generation: Once we generate the candidates, we must enrich them with features to produce the operable structures, as described in Section 3.3.3 (Sample Code 3.2). We start by retrieving the recently generated candidates (line 1). Then, we define the core (Section 3.4.1), which determines the tuple extraction algorithm (ShortestPathKernel) which implicitly defines its mandatory features. In this example, we add part-of-speech tags to the mandatory set of features (lines 6 and 7); however, we also included tokenization (line 4) and chunking (line 5) features, which are required to compute the part-of-speech tags. We then save this configuration (line 8), which we will use later during training and that we can use to produce operable structures from other candidate sentences for the *Charged* relation. Finally, we use the REEL StructureGenerator to produce the operable structures from the candidate sentences (line 9), and save them for later use (line 10).

Sample Code 3.2: Operable Structure Generation.

```
1  Set<CandidateSentence> cand = (Set<CandidateSentence>)SerializationHelper.read("train.ser");
2  StructureConfiguration conf = new StructureConfiguration(new ShortestPathKernel());
3  FeatureGenerator<SequenceFS<Span>> tok = new OpenNLPTokenizationFG("modelT.bin");
4  FeatureGenerator<SequenceFS<Span>> fgCh = new EntityBasedChunkingFG(tok);
5  FeatureGenerator<SequenceFS<String>> fgChSt = new SpansToStringsConvertionFG(fgCh);
6  FeatureGenerator<SequenceFS<String>> fgPOS = new OpenNLPPartOfSpeechFG("modelPOS.bin", fgChSt);
7  conf.addFeatureGenerator(fgPOS);
8  SerializationHelper.write("conf.ser", conf);
9  Set<OperableStructure> trD = StructureGenerator.generateStructures(cand, conf);
10 SerializationHelper.write("optr.ser", trD);
```

Relation Extraction Training: As described in Sections 3.4.1 through 3.4.3, we can use the operable structures to learn the relation extraction model, to extract tuples from text documents, and to perform a thorough evaluation of the relation extraction system. We now illustrate how REEL handles these operations.

For training, we load the operable structures, which are required to be labeled[3], and learn a relation extraction model (Sample Code 3.3). Specifically, we load the definition of the relation that we created during candidate generation (line 1), the configuration of the features (line 2), and the operable structures (line 3) both defined during operable structure generation. Next, we create an engine (line 5), which we described in Section 3.4.1. The engine includes the learning algorithm to train the relation extraction model from the operable structure and thus, must support its internal structure (e.g., kernel), as discussed. REEL allows users to define their own engines using different machine learning toolkits. Finally, we train (line 6) and save the learned model (lines 8-9), which we can use for tuple extraction and evaluation, as we see next.

[3]The labels of the operable structures can be loaded from the input text collection (e.g., from the ACE 2005 collection [Wal06]) or manually annotated.

Sample Code 3.3: Relation Extraction Training.

```
1 Set<RelationshipType> rTs = (Set<RelationshipType>)SerializationHelper.read("rTypes.ser");
2 StructureConfiguration conf = (StructureConfiguration)SerializationHelper.read("conf.ser");
3 Set<OperableStructure> trD = (Set<OperableStructure>)SerializationHelper.read("optr.ser");
4 //The engine is responsible for the training
5 Engine eng = new JLibSVMBinaryEngine(conf, rTs);
6 Model svmM = eng.train(trD);
7 //Finally, we can store the model in order to use it later
8 String modF = "CHARGEDModel.svm";
9 SerializationHelper.write(modF, svmMo);
```

Tuple Extraction: For tuple extraction (Sample Code 3.4) we employ all the capabilities described thus far. We first load the recently learned model and splitter (lines 1-3) that we then use to define our relation extraction system (line 4). This relation extraction system, which is provided in REEL, enables users to directly plug in their learned models and have a fully functional relation extraction system that can be used in application building settings. In addition to the model, the relation extraction system receives a text splitter (line 3) that defines the scope of the extraction (e.g., sentences), just as we did for candidate generation. To put the system to work, we load the documents from which we want to extract tuples (lines 5-8). (In this example we use the same type of dataset for training than we do for testing but we could easily plug in any other dataset and loader.) Finally, we iterate over the documents and print the extracted tuples (lines 9-11). The extractTuples method, provided in the relation extraction system, hides the complexity of processing the given document to obtain its operable structures and perform the extraction task.

Sample Code 3.4: Tuple Extraction.

```
1  String modF = "CHARGEDModel.svm";
2  Model svmM = (Model)SerializationHelper.read(modF);
3  OpenNLPMESplitter spl = new OpenNLPMESplitter("en-sent.bin");
4  ClassifierBasedRelationshipExtractor ext = new ClassifierBasedRelationshipExtractor(svmM, spl);
5  Set<RelationshipType> rTs = (Set<RelationshipType>)SerializationHelper.read("rTypes.ser");
6  Loader l = new MyLoader(rTs);
7  File AD = new File("/test/");
8  Dataset<Document> col = new Dataset<Document>(l, AD, false);
9  for (Document d : col) {
10   System.out.println(ext.extractTuples(d));
11 }
```

The output of the code above for a document that includes the text excerpt *"John Lennon's killer, Mark Chapman, was sentenced in 1981 to 20 years to life in prison after pleading guilty to second-degree murder"* in our running example will look like:

```
Charged[Person(Mark Chapman); Charge(second-degree murder); Date(1981)]
```

Relation Extraction Evaluation: Finally, for evaluation (Sample Code 3.5), we require the learned model that we want to evaluate, as well as the labeled operable structures that will represent the ground truth. We start by loading the learned model (lines 1-2) as well as the labeled operable structures (lines 3-9). We then define the evaluator (line 10), which we described in Section 3.4.3, and the performance measures that we will consider in the evaluation. Here, we measure our model using recall, precision, and F-measure (lines 11-16), which are already implemented in REEL, although the user can incorporate other measures that can be used directly, as we explained. Finally, we invoke the printEvaluatorReport method (line 17), which outputs the recall, precision, and F-measure values, as requested.

Sample Code 3.5: Relation Extraction Evaluation.

```
1  String modF = "CHARGEDModel.svm";
2  Model svmM = (Model) SerializationHelper.read(modF);
3  List<String> tF = FileUtils.readLines("testfiles.ser");
4  List<OperableStructure> l = new ArrayList<OperableStructure>();
5  List<OperableStructure> oS;
6  for (String s : tF) {
7    oS = (List<OperableStructure>)SerializationHelper.read(s);
8    l.addAll(oS);
9  }
10 Evaluator eval = new Evaluator();
11 Measure rec = new Recall();
12 eval.addMeasure(rec);
13 Measure pre = new Precision();
14 eval.addMeasure(pre);
15 Measure f = new FMeasure(1.0);
16 eval.addMeasure(f);
17 eval.printEvaluationReport(l, svmM);
```

Calling printEvaluatorReport outputs the performance values, which can be reported in many different ways (e.g., by appending the values to a file or by printing to console). We illustrate an example of such an output here:

```
Recall: 0.76 - Precision: 0.65 - F-Measure: 0.70
```

In this section, we walked through the steps required to develop and evaluate a typical relation extraction system in REEL. As shown, the code needed to produce such an extraction system within REEL is simple and easy to understand. This makes REEL a powerful framework to enable the deployment and evaluation of relation extraction systems for both application building and research.

3.6 Conclusions

In this chapter, we introduced REEL, an open-source framework to easily develop and evaluate relation extraction systems. REEL provides end-to-end infrastructure to handle relation extraction as a classification task, and leverages powerful existing toolkits for both

text processing and machine learning subtasks. Moreover, REEL effectively addresses the complex requirements of relation extraction and helps developers and researchers produce simple and easy-to-understand source code for their relation extraction systems. As part of the REEL distribution—as open source under the General Public License Version 3 (GPLv3) license, at `http://reel.cs.columbia.edu/`—we have included ready-to-use relation extraction systems (e.g., [BM05b; BM05a]). We have also integrated several text processing and machine learning toolkits, to illustrate how to incorporate and leverage external algorithms and toolkits.

The main contributions of this chapter, and those that make REEL preferable to existing toolkits, are: (i) the effective decoupling of text processing and learning components, which enables the integration and re-usability of different extraction algorithms; and (ii) the support to easily define and automatic reinforce entity and relation constraints that can be shared across extraction tasks and systems. We have identified these contributions as crucial for the easy development, evaluation, and deployment of relation extraction systems during our interaction with REEL. In this dissertation, in particular, we used REEL to develop, evaluate, and select full-fledged relation extraction systems for our experiments. We provide details about the application of REEL in each individual section. Finally, despite its short lifespan, REEL has received increasing, substantial attention across the research community. We expect this trend to continue as well as to open new research and collaboration opportunities.

Chapter 4

Sampling Documents for Scalable Information Extraction

In Chapter 2, we described a state-of-the-art approach for deploying an information extraction system over large text collections and characterized the importance of collecting representative extraction task-specific document samples from the collections. As discussed, a document sample from a collection can be valuable, for instance, to help select and rank the collection documents for the extraction task: techniques such as QXtract [AG03], FactCrawl [BLNP12], or PRDualRank [FC11] use these samples to learn words and phrases that separate those documents that lead to the extraction of tuples for a relation of interest from those documents that do not, and should hence not be processed, for efficiency. Importantly, the samples on which these techniques rely must be collected in a collection-specific way, because the focus and language of each collection generally differs from those of other collections.

Despite the important role of sampling in the techniques above, the sampling approaches that they use are far from ideal, as we will see. Specifically, these techniques adopt flavors of sampling that rely on high-precision queries to target certain documents efficiently, but fail to capture the large variety of extraction-relevant document characteristics in the useful documents. Consequently, they miss important groups of documents during sampling, which other sampling strategies can effectively obtain, as we will show experimentally.

In this chapter, we systematically study the space of query-based document sampling techniques for information extraction. Specifically, we consider (i) alternative query execution schedules, which vary on how they account for the query effectiveness; and (ii) alternative document retrieval and processing schedules, which vary on how they distribute the extraction effort over documents. We conduct a large-scale and fine-grained experimental evaluation over real Web collections, and for a large variety of information extraction tasks, to assess the merits of the alternative query execution and document retrieval and processing strategies. We also explore several different query generation techniques, for robustness.

The conclusions of our study are twofold. Regarding query execution, schedules that focus on queries with a high fraction—and number—of useful documents, namely, the *effective* queries, improve sampling efficiency: These schedules require issuing a small number of queries and processing few documents to collect document samples of a certain size. In contrast, schedules that prioritize less-effective queries need to issue many (potentially diverse) queries to retrieve a desired number of useful documents, hence improving sampling quality: These schedules are likely to collect documents that cover distinct extraction-relevant document characteristics (e.g., words and phrases related to different natural disasters for our *Occurs-in* task) in the useful documents. Regarding document retrieval and processing, schedules that process the documents for each query exhaustively at once improve sampling efficiency when the sampling technique focuses on effective queries. In contrast, schedules that process documents incrementally and in rounds improve sampling quality, because a larger variety of documents—from a larger number of queries—is processed. As we will see, fundamentally different sampling techniques (i.e., with distinct implications in sampling efficiency and quality) are possible.

In short, the main contributions of this chapter are:

- A systematic study of query-based document sampling techniques for information extraction over text collections that considers (i) alternative query execution schedules and (ii) alternative document retrieval and processing schedules (Section 4.2).

- The first large-scale and fine-grained evaluation of query-based document sampling

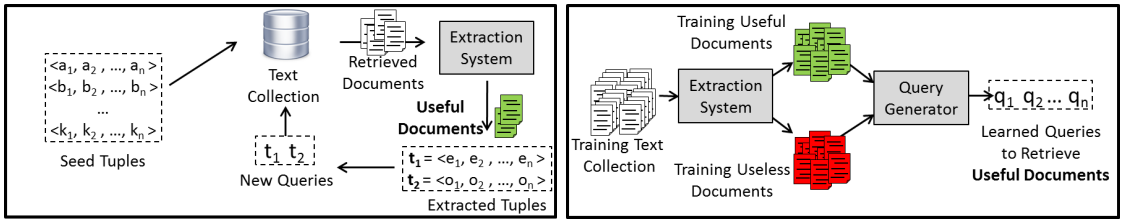
techniques for information extraction. We perform our experiments over real Web-accessible collections and for a large variety of extraction tasks. We show the implications in sampling efficiency and quality of different query execution schedules, as well as of different document retrieval and processing schedules (Sections 4.3 and 4.4).

We now review necessary background and define our problem of focus in this chapter.

4.1 Background and Problem Definition

Given an information extraction task, producing high-quality, representative document samples from a text collection is a challenging process, for two main reasons. *(1) Sampling efficiency:* the document sampling process has to be efficient and lightweight because, as discussed above, it is often used to make the overall information extraction execution over text collections efficient and scalable. This efficiency requirement is complicated by the fact that analyzing the documents, to decide the composition of the samples, is an expensive proposition because it often involves running the extraction system at hand on the documents. Furthermore, and particularly for deep web collections, document samples can only be collected, by definition, by querying the (remote) contents of the collections, which is expensive. *(2) Sampling quality:* the document sampling process has to return documents that represent the relevant extraction-related document characteristics in each collection. This quality requirement is complicated by the fact that useful documents for the information extraction task, are often a small minority of the collection documents, as discussed. Furthermore, even within a relatively small number of documents, the sampling process should capture the large variations in language and general content in the documents. Finally, document sampling techniques should be applicable to fully accessible text collections as well as to deep web text collections, which are only accessible via querying (Section 2.2).

Query-based document sampling has also been studied beyond information extraction, for other text-centric tasks. As notable examples, [CC01], [BYG08], and [ZZD11] developed document sampling techniques for the generation of generic descriptors of the collections. Unfortunately, these approaches are ineffective for our information extraction scenario, because they focus on obtaining random document samples. As we discussed above, our



(a) Bootstrapping-based useful documents retrieval. (b) Learning-based useful documents retrieval.

Figure 4.1: Two main families of existing query generation techniques for useful document retrieval.

scenario requires that the document samples represent the often small minority of documents that lead to extraction output for a given information extraction task. To sufficiently characterize the documents in such small portions of the collections through random sampling, the above techniques would require issuing an exorbitant number of queries to the collections.

Based on the discussion above, the problem of focus in this chapter is that of efficiently collecting high-quality document samples for information extraction from text collections, as follows:

Problem Definition 1 *Consider a text collection C and an information extraction system E trained to extract tuples for a relation from text. To enable efficient and effective information extraction over collection C, we need a sample of documents from C that represents the population of useful documents in C with respect to E. Specifically, the goal is to obtain a sample of useful documents that satisfies certain quality metrics (e.g., diversity in the tuples extracted with E from the sampled documents) while satisfying certain efficiency-related requirements (e.g., minimize the number of documents processed with E and the number of queries issued to C as part of the sampling process).*

Existing query-based techniques for retrieving useful documents from a collection fall into two families. Techniques in the first family adopt a bootstrapping approach: Starting with a small number of "seed" tuples for the relation of interest, these techniques iteratively retrieve (potentially useful) documents by issuing as queries the seed tuples and, later, the new tuples that the extraction system discovers from documents as they are retrieved (Figure 4.1a). Earlier efforts to address the efficiency and scalability of the ex-

traction process (e.g., QXtract [AG03], FactCrawl [BLNP12], and PRDualRank [FC11]) have adopted this family of techniques in their sample generation step, because queries tend to be high-precision. Unfortunately, as we will show experimentally, these techniques compromise recall and often miss important relevant groups of useful documents, which is undesirable during the sampling step.

Techniques in the second family adopt a statistical learning approach that aims to alleviate the recall limitation above: Using a training sample of useful and useless documents labeled "for free" with the information extraction system at hand, these techniques learn keywords and phrases that are discriminative of the useful documents (Figure 4.1b). Importantly, the learned keywords and phrases often include a score that roughly corresponds with their expected precision and recall for useful documents. These scores can be systematically exploited when issuing these learned keywords and phrases as text queries to retrieve potentially useful documents. For instance, QXtract [AG03] issues the queries in descending score order, to first process queries that are likely to retrieve useful documents with high recall and precision. QXtract processes the documents retrieved by each query exhaustively at once before processing those retrieved by the following query. Unfortunately, these techniques mainly tackle the efficiency of the extraction process, one of the crucial aspects in our sampling problem. As we will see, to also address the sampling quality we need to choose carefully both the query execution as well as the document retrieval and processing strategies.

In the next section, we discuss different query execution and document retrieval and processing strategies, along with their implications in sampling efficiency and quality. We in turn introduce several different sampling techniques, which we evaluate experimentally in later sections.

4.2 Document Sampling Strategies

We now systematically study query-based document sampling techniques for information extraction over a text collection. We focus on learning-based methods, which rely on a learned set of text queries to retrieve potentially useful documents for an information ex-

traction task of interest, as discussed in Section 4.1. Unlike in the existing literature, though, we consider tackling both sampling quality and efficiency. We start by outlining—and analyzing the efficiency and quality of—different alternatives for processing the (learned) set of queries and their retrieved documents, namely, the *query–document space* of the queries (see Figure 4.2). We then discuss how we can exploit the information that we gather from each query along the sampling process (e.g., the number of useful and useless documents that the query returns) to improve different aspects of the process. In turn, we introduce the sampling techniques that we study in this chapter, which we evaluate experimentally in Sections 4.3 and 4.4.

4.2.1 Exploring the Query–Document Space

We now consider different alternatives to exploring the query–document space of a set of queries for our sampling problem. We first consider alternative query execution schedules, which vary on how they account for the query effectiveness. Specifically, for a pool of documents retrieved by a query, we define the *effectiveness* of the query as the fraction of useful documents within this document pool.[1] Then, and in an orthogonal dimension, we consider alternative document retrieval and processing schedules, which differ on how they determine the document pool retrieved by each query and how they distribute the extraction effort over documents. We discuss these alternatives in detail next.

Query Execution: The order in which we process queries during sampling, namely, the *query execution order*, is crucial to the efficiency and quality of the sampling process. For efficiency, on one hand, we need to prioritize effective queries (i.e., the queries that retrieve useful documents with high precision and recall), so that we mainly process—hence sample—useful documents. This is motivated by the fact that the sampling cost is a function of the number of issued queries—necessary to retrieve documents for the sample—and the number of documents retrieved and processed—necessary to decide the composition of the sample. The approach in [AG03], for example, approximates this query order based on the learned query scores: This approach uses the learned score of a query as a surrogate

[1] More formally, the effectiveness of a query is based on the so-called precision@K in information retrieval, where relevance is defined in our case as usefulness and K is the number of documents to process.

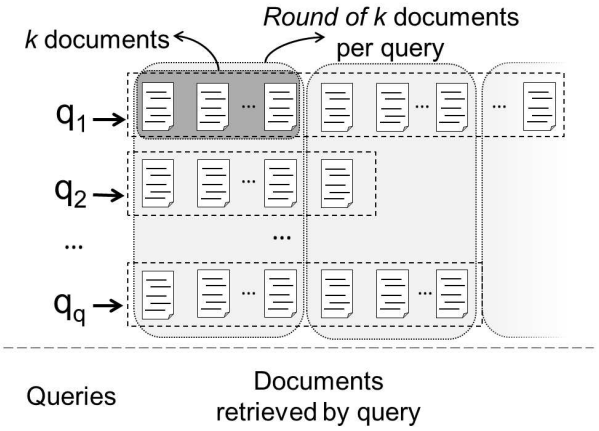


Figure 4.2: Query–document space.

of its effectiveness and arranges the queries in descending score order. Figure 4.3 shows an example of such query order for *Occurs-in*: the (top) query [earthquake] is more effective than query [richter], because it retrieves more useful documents for the same number of processed documents.

Processing queries in decreasing effectiveness order leads to efficient executions that identify a sample of useful documents quickly. Unfortunately, if the query execution process is only guided by efficiency, the overall sampling quality might suffer. To see why, consider once again the example in Figure 4.3. Specifically, if the query execution process were to focus, say, on queries [earthquake] and [richter], which are highly effective for *Occurs-in*, we would likely produce a document sample whose useful documents are predominantly about earthquakes and not about other natural disasters that should be included in the sample as well.[2]

We thus argue that for quality we should sometimes prioritize less-effective queries, so that a larger—hence potentially more diverse—set of queries needs to be processed to obtain a desired number of useful documents. In our example in Figure 4.3, for instance, such a query execution order would process query [aftermath] before processing other more effective queries (e.g., queries [vortex] or [earthquake]) and, more importantly, it would be more likely

[2]Additionally, note that if query [earthquake] retrieves documents including the word "Richter," query [richter] may lose effectiveness, because many of the useful documents that query [richter] returns may have already been processed. We study how to stop processing ineffective, underperforming queries in Section 4.2.2.

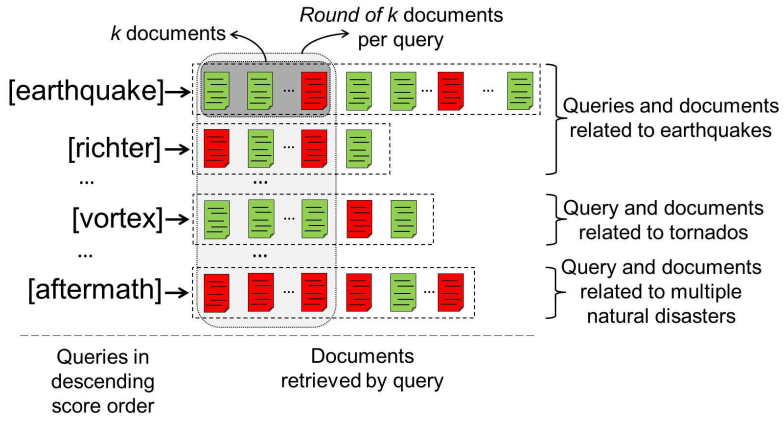


Figure 4.3: Query–document space of a set queries for the *Occurs-in* relation. Useful and useless documents are illustrated in green and red, respectively.

to cover documents about earthquakes, tornadoes, as well as other natural disasters, because a larger number of queries would be processed. This quality-driven approach, however, is problematic for two reasons. First, arranging the queries in such query order, or an approximation thereof, is nontrivial, unlike with the efficiency-driven query execution order above. Second, following this query execution order might compromise sampling efficiency dramatically, because many useless documents would need to be processed to retrieve a desired number of useful documents. Next, we discuss how different document retrieval and processing strategies can help address these limitations.

Document Retrieval and Processing: In addition to query execution, the strategy we adopt to retrieve and process the documents during sampling, namely, the *document retrieval and processing strategy*, is also crucial to the efficiency and quality of the sampling process. A possible choice is, of course, to process the documents returned by each query exhaustively at once, as suggested in [AG03]. Importantly, such an exhaustive strategy would promote the efficiency and quality considerations of the adopted query execution approach: If, for instance, the query execution is guided by efficiency (i.e., effective queries are prioritized), as in [AG03], exhaustively processing the documents returned by each query will yield efficient sampling executions, because the number of queries to issue and documents to process to collect a desired number of useful documents will be relatively small. Analogously, if the query execution is guided by quality (i.e., it prioritizes less-

effective queries), processing all documents returned by each query would produce high-quality sampling executions, because a larger, potentially more diverse set of queries will be processed. Unfortunately, by promoting the considerations of the adopted query execution, this exhaustive document processing strategy would also preserve their discussed quality and efficiency limitations.

An alternative document retrieval and processing strategy, and one that would alleviate the limitations of the exhaustive strategies above, would be to process the documents returned by each query iteratively and in rounds. Specifically, for a given query execution order, this strategy would iterate over the queries in order, processing only a certain number of documents per round. In Figure 4.3, we identify the documents in the first round of an iterative strategy that processes k documents from each query per round (see lightly shaded area in Figure 4.3). As a result of this iterative process, documents will potentially be sampled from larger sets of queries—hence addressing sampling quality—and the extraction effort will be evenly distributed among queries during each round—hence addressing sampling efficiency. To better illustrate this, consider again the lightly shaded documents from our *Occurs-in* example in Figure 4.3: These documents form a rather diverse sample—with documents about earthquakes, tornadoes, and other disasters—and only a fraction of the (many) useless documents retrieved by less-effective queries (e.g., query [aftermath]) need to be processed during the first round.

Despite the advantages of the iterative strategy above, specifying a number of documents per round that suitably balances efficiency and quality is a difficult proposition: Large values for such number would exhibit similar limitations to those of the exhaustive approach discussed above, while small values would affect sampling efficiency drastically due to the high querying cost that would be incurred. We experimentally evaluate the efficiency and quality implications of the choice of the number of documents to process per round in Sections 4.3 and 4.4. An additional problem of using small values is that we would be unable to precisely measure the real effectiveness of queries, a crucial measurement, as we discuss next.

4.2.2 Exploiting Observed Information

So far, we have discussed the query–document space exploration as a static, once-and-for-all process. However, there is valuable information (e.g., the number of useful and useless documents that a query returns) that we can gather gradually, as the sampling process progresses, and that we can use to improve this process. We now discuss how we can exploit this information (i) to revise the query execution order, for sampling efficiency and quality; and (ii) to filter underperforming queries, for sampling efficiency:

Revising Query Execution Order: The learned score of a query is often used as a surrogate of its effectiveness, as argued earlier in this section, so we can expect the query order given by these scores to be correlated with that of the real effectiveness of the queries. However, for a given collection, these two query orders may differ considerably (e.g., due to the contents of the collection or the indexing and retrieval techniques thereof), and hence the query execution order may have to be revised. For instance, in our *Occurs-in* example in Figure 4.3, prioritizing query [vortex] would yield more efficient sampling executions than processing the documents in descending score order.

Fortunately, exhaustively processing the documents returned by a query to effectively measure its effectiveness is unnecessary: We can in fact gauge the real effectiveness of a query by only processing a relatively small subset of its returned documents, because the fraction of useful documents is expected to remain largely stable across its retrieved documents [IAJG07].[3] For instance, in our example in Figure 4.3, we could process the first k documents returned by each query, to conclude that queries [vortex] and [aftermath] are, respectively, the most and least effective queries, and revise the query execution order in light of the observed information.

Filtering Underperforming Queries: By definition, there are two operations during sampling that hurt sampling efficiency, namely, issuing queries to the collection at hand that retrieve none—or a low fraction of—useful documents and retrieving and running the information extraction system of choice over a useless document. We argue that, for efficiency—and at the expense of a modest lost in quality—we can exploit the gauged

[3] This idea of "probing" queries to estimate their effectiveness is used in a preprocessing step in [BLNP12] for the related problem of ranking documents to improve the efficiency of the extraction process.

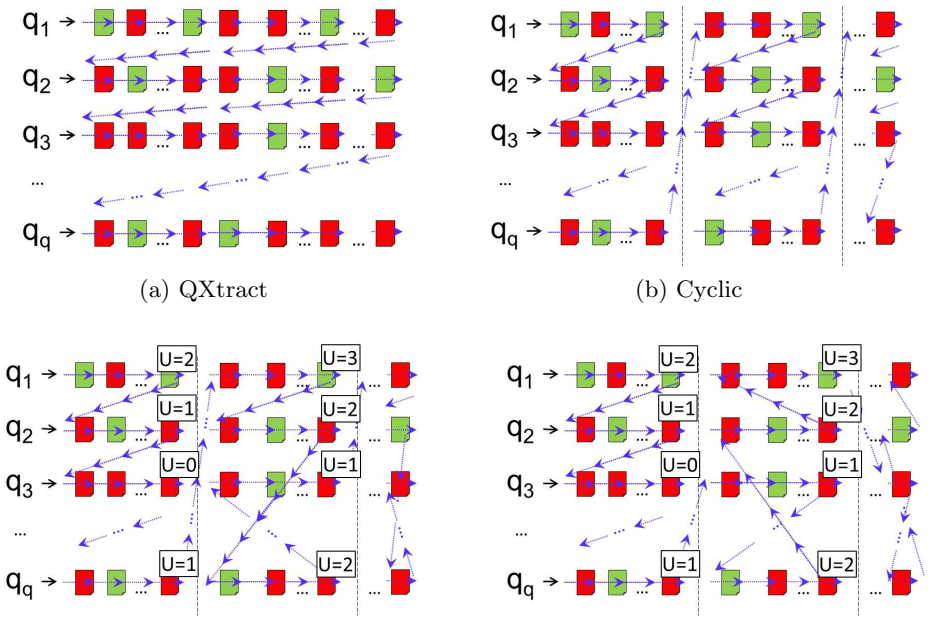
effectiveness of queries to avoid such (undesirable) cases and, in effect, focus only on cost-effective queries. For instance, in our example for *Occurs-in* in Figure 4.3, if we filtered query [aftermath], we would avoid a considerable extraction effort—over multiple useless documents—at the expense of losing one useful document.

Based on the discussion above, we consider applying two filtering options. Our first alternative avoids issuing a query altogether if the observed effectiveness of previously issued queries is below a certain threshold. This filtering scheme is possible when we initially issue queries in descending score order, because the performance of the queries is expected to drop as a function of their order. In Figure 4.3, for our *Occurs-in* example, we may filter query [aftermath] if previous queries exhibited poor effectiveness. Our second alternative filters already issued queries whose real, observed effectiveness drops below a certain threshold, to avoid processing useless documents. For instance, if we decide to filter queries that do not retrieve useful documents within the first k documents, the documents beyond k returned by query [aftermath] in Figure 4.3 for *Occurs-in* would not be processed. Of course, deciding the settings for these filtering conditions is challenging, and we consider several options in Sections 4.3 and 4.4 together with their impact on sampling efficiency and quality.

4.2.3 Sampling Techniques

So far, we have discussed the components involved in query–document space exploration as well as explained how we can exploit observed information to adaptively revise the query execution order and to focus the sampling effort. We now define the (arguably) most interesting query-based document sampling techniques for information extraction over a text collection, which we summarize in Table 4.1. As we will see in our experimental evaluation, we focus on techniques that collect high-quality document samples while keeping the sampling cost at reasonable levels. For the completeness of our evaluation, however, we include other sampling techniques, which we describe in the next sections, as needed. Importantly, some of the techniques in Table 4.1 (e.g., QXtract [AG03]) have been introduced in the literature whereas others have not. We list them all here, to assess their merits and limitations experimentally later in Sections 4.3 and 4.4.

QXtract: *QXtract* [AG03] explores the query–document space by issuing queries in de-



(a) QXtract (b) Cyclic

(c) Opportunistic (d) Balanced

Figure 4.4: Examples of query–document space exploration strategies. Useful and useless documents are illustrated in green and red, respectively.

Name	Query Execution	Document Retrieval and Processing	Query Order Revision	Query Filtering
QXtract	>	→	-	-
Cyclic	>	↻	-	-
Opportunistic	>	↻	+	-
Balanced	<	↻	+	-
F–QXtract	>	→	-	+
F–Cyclic	>	↻	-	+
F–Opportunistic	>	↻	+	+
F–Balanced	<	↻	+	+

Table 4.1: Sampling techniques and the alternatives they consider for each relevant aspect. For query execution, we consider prioritizing effective queries (>) or less-effective queries (<). For document retrieval and processing, we consider processing documents exhaustively at once (→) or iteratively and in rounds (↻). We finally consider techniques that perform query order revision or query filtering (+) and techniques that do not (-).

scending learned score order and processing the documents retrieved by each query exhaustively at once (Figure 4.4a). QXtract produces relatively efficient sampling executions;

however, it may compromise sampling quality, as discussed earlier in this section.

Cyclic: *Cyclic* explores the query–document space by issuing queries in descending learned score order and processing the documents retrieved by each query iteratively and in rounds (Figure 4.4b). Cyclic addresses the sampling quality deficiencies of QXtract above, because it requires issuing a larger—hence potentially more diverse—set of queries to retrieve a desired number of useful documents. For instance, to collect three useful documents in Figure 4.4b, Cyclic processes the documents returned by two queries, namely, q_1 and q_2, whereas QXtract processes the documents returned by one query, namely, q_1.

Opportunistic: *Opportunistic* explores the query–document space by prioritizing—and issuing—effective queries and processing the documents retrieved by each query iteratively and in rounds (Figure 4.4c). Opportunistic initially prioritizes queries according to the learned score; then, between rounds, and as it gathers relevant information for each query, Opportunistic revises the query execution order using the real, observed effectiveness of queries. The sampling quality of Opportunistic may suffer, though, because some groups of documents may still be underrepresented. To see why, consider Figure 4.4c: If the sampling process stops after collecting five useful documents (i.e., during the second round of documents retrieved by q_1), q_1 will contribute three useful documents to the sample whereas q_2, q_3, and q_q will contribute at most one useful document each.

Balanced: *Balanced* explores the query–document space by prioritizing—and issuing—less-effective queries and processing the documents retrieved by each query iteratively and in rounds (Figure 4.4d). Because finding an initial query order for the queries is problematic, as discussed, Balanced initially issues queries in descending score order; then, between rounds, and as it gathers relevant information for each query, Balanced revises the query execution order using the real, observed effectiveness of queries. By prioritizing less-effective queries, Balanced alleviates the quality limitation of Opportunistic above. Specifically, if in Figure 4.4d we stop after collecting five useful documents (i.e., during the second round of documents retrieved by q_3, which will be now prioritized), each query will contribute a similar number of useful documents to the sample.

The techniques described thus far do not include the filtering step described earlier in this section. We define variants of these techniques that incorporate query filtering, which

we refer to as *F–QXtract*, *F–Cyclic*, *F–Opportunistic*, and *F–Balanced*, respectively (see Table 4.1). These filtered techniques run as their unfiltered counterparts, and decide the queries to filter based on the filtering options described earlier in this section. Next, we describe the settings for our in-depth experimental evaluation of sampling techniques for information extraction.

4.3 Experimental Settings

We describe the details of our experimental evaluation of the query-based document sample generation techniques for information extraction.

Web Collections: We collected a representative set of 335 real Web collections across different topics by following an approach similar in spirit to that of [GIS03] over the Open Directory Project directory [ODP15]: We first selected the 8 categories with the highest number of entries, namely, Business, Society, Arts, Science, Computers, Recreation, Shopping, and Sports. From each category, we then selected the 5 most popular subcategories along with their corresponding 5 most popular subsubcategories, for a total of 200 subsubcategories. We then randomly chose 7 unique Web collections with a text search interface from each subsubcategory.[4] Finally, we randomly selected 335 collections from this set of collections, which we split into a tuning set (48 collections, or 15% of the collection set) and a test set (287 collections, or 85 % of the collection set). We report our results over the test set.

Interaction with Web collections: We developed an end-to-end system to support the large number of Web collections in our experiments (see above) and to automatically query the real contents of the collections. For our experiments, we focused on HTML-based Web collections, which were the vast majority at the time of our experiments. Specifically, our system consists of three main components, namely, the *learning*, the *querying*, and the *maintenance* components, as follows:

- *Learning component:* For a given URL (e.g., `http://www.fema.gov`) of a collection, the learning component determines how to retrieve documents from the collection

[4] For each subsubcategory with fewer than 7 entries, we selected all its collections.

via querying. The learning component starts by finding the search interface of the collection using the decision tree in [MCSSMTLA13] to, in turn, identify the querying protocol (e.g., GET or POS) as suggested in [MKK+08]. After identifying the search interface, the learning component detects the HTML component with the returned documents in the results page. For this, we use an approach similar in spirit to that in DeLA [WL03], which is based on the observation that the component where returned documents are placed tends to vary substantially across result pages for different queries. For two given result pages, our technique uses the Hiperfingladal index [oJC97] to identify the component with the highest concentration of changes. We obtain result pages by submitting 50 queries obtained from the main page of the collection [She09], which makes them likely to retrieve documents from the collection. We also submit 5 queries that are unlikely to retrieve any documents (e.g., a sequence of random characters), to obtain "empty" result pages. Finally, for queries that return more documents than those visible in a result page, the learning component finds the component for navigational links (e.g., links or images with "next," "more results," ">," or numbers). We do this by identifying HTML components that include the same text (e.g., "next") but different links (e.g., links that include the query text) across result pages, and also by verifying that the linked URL is similar to other result pages based on the tree edit distance of the HTML structure of the pages [FB11]. We obtain a similarity threshold as the lowest similarity value across all 55 result pages from the queries above.

- *Querying component:* The querying component obtains the documents that a query returns. Specifically, the querying component uses the output of the learning component above to issue a query and navigate its result pages, and extracts (hyperlinks to) returned documents from the result pages. We obtain the clean text of each result document using Tika [Tik15], a toolkit for extracting text from various file types.

- *Maintenance component:* The maintenance component keeps our system up-to-date when the available collections experience format changes (e.g., different formatting of pages or new placement of advertisements). For this, we use the tree edit distance-

Relation	Useful Documents (%)	
	SSK	BONG
Person–Career	56.20%	55.95%
Natural Disaster–Location	2.03%	2.74%
Man Made Disaster–Location	0.80%	0.87%
Person–Travel Destination	1.08%	4.67%
Person–Charge	1.55%	1.84%
Election–Winner	0.24%	0.84%

Table 4.2: Relations for our experiments along with fraction of useful documents in TREC 1-5 collections. In this table, Travel Destination and Winner are of type Location and Person, respectively.

based approach in [FB11]. We regularly compare the structure of the latest observed versions of the pages of interest (e.g., search forms, result pages, result documents) against local copies of these pages. We set the validation threshold to 1 for the search forms, so that we detect all changes in the search interfaces. Our system updates by re-running the learning phase when the similarity of a given page and its local counterpart is below the given threshold for the page.

Importantly, because HTML pages often exhibit broken schemas and cannot be automatically parsed, we clean all HTML pages that our system retrieves with the htmlcleaner tool [htm15].

Training Collection: To learn the queries for our sampling strategies, we need a text collection that includes useful documents for the extraction tasks of interest, as discussed in Section 4.1. For this purpose, we combined all documents in the TREC 1-5 collections [TRE00] to form a collection of 1,038,957 unique documents.

Entity and Relation Extraction Systems: To include a variety of extraction approaches, we considered different relation extraction systems for each relation (see next), as well as different entity extraction systems for their corresponding entities. For relation extraction systems, we selected the two best performing combinations via 5-fold cross validation over a set of manually annotated documents. Likewise, for entity extraction systems, we selected the best performing combination for each entity type, and used it across all extraction tasks. However, for diversity, whenever we had ties in performance, we selected the (arguably) less common contender. We provide details next:

- *Relation Extraction:* To extract our relations, we trained relation extraction systems using REEL (see Chapter 3). The two best performing systems, and the ones that we use in our experiments, are Subsequence Kernel [BM05b] (SSK) and Bag of n-grams Kernel [GLR06] (BONG).

- *Entity Extraction:* To extract *person* and *location* entities, we used the StanfordNLP named entity tagger [Sta15b]; for other entities, we trained our own entity extractors using E-txt2DB [Etx12]. Our final models are Maximum Entropy Markov Models [MFP00] for *natural disasters* and Conditional Random Fields [ML03] for the remaining entities.

Relations: Table 5.2 shows the broad range of relations from different domains that we extract for our experiments. We also include the fraction of useful documents for each relation in our training collection for the different extraction systems above. Our relations include *sparse* relations, for which a relatively small fraction of documents (i.e., less than 2% of the documents) are useful, as well as *dense* relations.

Bootstrapping-based Sampling Techniques: In addition to the techniques discussed in Section 4.2, we evaluate the bootstrapping-based approach proposed in [AG03]—and described in Section 4.1—that derives queries from all attributes in extracted tuples. We also experiment with queries derived from each attribute individually, as done in [FC11], to assess their impact in sampling quality and efficiency. The bootstrapping-based techniques that we explore are defined as follows:

- *Tuples* [AG03] uses all tuple attributes in the query. For example, for the *Occurs-in* relation, Tuples produces the query [adairsville AND tornado] from the tuple ⟨adairsville, tornado⟩.

- *P-Tuples* [FC11] uses the most "specific" (see below) tuple attribute of the relation in the query with the goal of producing high-precision queries. To determine the most specific attribute in a relation, we analyze the schema of all relations supported by OpenCalais [Ope15a], an online service for information extraction, and use the least common relation attribute. In our *Occurs-in* relation, for instance, P-Tuples

uses the *natural disaster* attribute, because this is the attribute that appears in the fewest OpenCalais relations, specifically in just one relation out of 83. P-Tuples thus produces the query [tornado] from the tuple ⟨adairsville, tornado⟩.

- *R-Tuples* [FC11] uses the most "general" (see below) tuple attribute of the relation in the query with the goal of producing high-recall queries. To determine the most general attribute in a relation, we analyze the schema of all relations supported by OpenCalais and use the most common relation attribute. In our *Occurs-in* relation, for instance, R-Tuples uses the *location* attribute, because this is the attribute that appears in the most OpenCalais relations, specifically in 16 relations out of 83. R-Tuples thus produces the query [adairsville] from the tuple ⟨adairsville, tornado⟩.

As discussed in Section 4.1, given a collection, bootstrapping-based techniques start with a small seed of tuples for the relation of interest likely to be mentioned in the collection. We rely on a fully automatic approach to obtain such tuples: We collected 20,000 unique documents from each collection using the crawling technique by Barbosa et al. [BF10]. The technique in [BF10] generates initial queries using words in the main page of the text collection, and subsequently generates more queries using frequent keywords from the documents retrieved using the initial queries. We then run our information extraction systems over the crawled documents, to obtain tuples for each collection–information extraction system pair. We do not consider the cost of obtaining these tuples in the overall sampling cost reported in Section 4.4, to focus on quantifying the actual cost of sampling. For collections that did not produce tuples following this strategy, we generated seed tuples from the training collection.

Learning-Based Query Generation Techniques: We now describe different query generation techniques that learn queries from a training document sample, as discussed in Section 4.1. Query generation techniques rely on two building blocks, namely, the *candidate set of keywords* and the *query generation algorithm*. The candidate set of keywords specifies the words (e.g., all words except for stopwords) in the training documents that the query generation algorithm can use to construct queries. The query generation algorithm automatically learns as text queries discriminative words and phrases that separate useful

from useless documents. As described in Section 4.1, these query generation techniques assign a score to each word or phrase, which is generally a function of its precision and recall for useful documents. In detail, our candidate sets of keywords and query generation techniques are as follows:

Candidate Set of Keywords: We study two candidate set of keywords. For our first set, we removed: (i) English stopwords reported in MySQL, as they are not effective as queries and (ii) rare words (i.e., words that appeared in less than 0.003% of the training documents) and frequent words (i.e., words that appeared in more than 90% of the training documents). For our second set, we also removed words in tuple attributes (e.g., "tornado"), as originally suggested in [AG03]. We refer to the first candidate set of keywords as *explicit*, since attribute values can be used to construct queries; accordingly, we refer to the second candidate set of keywords as *implicit*.

Query Generation Algorithm: We explored several techniques from two fundamentally different approaches: (i) *keyword selection*, which produce single-keyword queries from words that effectively separate useful from useless documents; and (ii) *keyword combination*, which produce phrase queries (e.g., ["tornado swept"]) or Boolean queries (e.g., [tornado AND vortex]) from word combinations that are discriminative of the useful documents. Specifically, we evaluated three keyword selection techniques (SVM, IG, and χ^2) and two keyword combination techniques (Ripper and SP), which effectively cover existing query generation algorithms in the literature. We provide a brief description of these techniques, and explain how they score words and phrases:

- *SVM* [MBGM04] trains a linear support vector machine classifier [Joa98a] using the candidate set of keywords as Boolean features, and scores them with their corresponding learned weights.

- *IG* [MBGM04] scores each keyword in the candidate set with its *information gain* value [KL51].[5] We ignore keywords that are more frequent in the useless documents than in the useful documents.

[5]The information gain of a keyword W is defined as $IG(C) = H(C) - H(C|W)$, where $C = \{\text{useful}, \text{useless}\}$, and $H(C)$ and $H(C|W)$ are the entropy and the conditional entropy, respectively.

- *Chi-Squared* (χ^2) performs the Pearson's χ^2 test [Pea00] over the candidate set of keywords and scores them with their corresponding χ^2 value. Because the test runs over a 2×2 contingency table for each keyword—with usefulness of documents (i.e., useful or not) and occurrence of a keyword (i.e., it occurs in the document or not)—and because the table observations that we obtain from the training sample are rather small, we apply Yates's correction [Yat34] to the observations[6]. Yates's correction alleviates the upward bias of Pearson's χ^2 test in 2×2 contingency tables with low observations.

- *Ripper* [AG03] uses the Ripper algorithm [Coh95] to generate classification rules consisting of combinations of words that define useful documents. The algorithm in [AG03] then transforms the rules into Boolean conjunctive queries. For example, the rule <"vortex" AND "wind" $\Rightarrow$ useful> is transformed into the Boolean conjunctive query [vortex AND wind]. A query is scored with its expected precision, defined as the ratio of useful documents to the total of documents in the training set that match its original rule.

- *Significant Phrases* (SP) [BLNP11a; BLNP12] learns the most frequently collocated pairs of words [Dun93] from the useful documents and reports them as phrase queries. For example, for the *Occurs-in* relation, SP produces queries such as ["richter scale"] and ["snow storm"]. SP scores each phrase with the Pearson's χ^2 value computed over its keywords, which indicates how independent its keywords are from one another. To guarantee that the queries (i.e., collocated pairs of words) are real phrases in the document sample, we generate all phrases and remove those that do not comply with the candidate set of keywords at hand: (i) for explicit, we remove phrases with only stopwords, rare, or frequent words; (ii) for implicit, we also remove phrases that include words in the attribute values.

We used Weka 3.6 [HFH+09] with default settings to implement SVM (SequentialMinimalOptimization), IG (InfoGainAttributeEval), χ^2 (ChiSquaredAttributeEval with Yates

[6]The corrected χ^2 value is obtained from $\chi^2(K) = \sum_{i \in \{0,1\}} \sum_{C \in \{+,-\}} \frac{((\mathcal{O}^K_{i,C} - \mathbb{E}^K_{i,C}) - 0.5)^2}{\mathbb{E}^K_{i,C}}$, where k is the keyword, $\mathcal{O}^K_{i,C}$ and $\mathbb{E}^K_{i,C}$ are the observed and the expected value for K of the contingency table, respectively, and i and C index the occurrence of the term and the usefulness of a document, respectively.

correction), and Ripper (JRip). To implement SP, we used the significant phrases implementation of LingPipe [lin15] with default settings, as suggested in [BLNP11a].

Sampling Techniques: We evaluate the techniques described in Section 4.2 and the bootstrapping-based techniques described above. For QXtract, we retrieve and process 1000 documents per query, while we consider different numbers of retrieved documents for Cyclic, Balanced, and Opportunistic. Also, for reference, we compare a sampling technique that prioritizes less-effective queries from the ground up (i.e., without previously assessing the real effectiveness of queries). Specifically, this technique, which we refer to as *Reverse*, proceeds as QXtract (see Section 4.2), although processing top-Q queries in ascending score order. We use different values of Q in our experiments.

Filtering Conditions: We rely on two filtering conditions, which correspond to the alternatives described in Section 4.2. The first filtering condition stops processing queries based on the performance of the latest N queries that were issued. Specifically, we stop processing queries when, out of these N queries, the fraction of queries that retrieve at least one useful document is below a certain threshold τ_r (see (1) in Table 4.3). The main impact of this filtering condition occurs during the first query round because, as discussed, queries are initially issued according to their effectiveness. The second filtering condition stops processing queries based on their actual performance, as follows: We stop processing a query q if its effectiveness computed over the last M retrieved documents (i.e., the precision@M of the query) is below a certain threshold τ_q (see (2) in Table 4.3). We evaluated different values for the parameters in these conditions: We varied $N \in [10, 100]$, $\tau_r \in [0.02, 0.25]$, $M \in [5, 50]$, $\tau_q \in [0.05, 0.25]$. Finally, we kept for each strategy the settings that collected on average the largest and highest-quality samples for the same sampling cost, which we summarize in Table 4.3.

Sampling Execution and Termination: We let each sampling execution issue at most 500 unique queries and process at most 10,000 unique documents, to keep the sampling cost to reasonable levels. Given an information extraction system and a collection, the output document sample includes all useful documents for the extraction task at hand that are processed along the sampling execution. We also terminate the sampling process after collecting 400 useful documents. According to the results over our tuning collections,

Technique	Filtering Condition			
	(1)		(2)	
	τ_r	N	τ_q	M
F-QXtract	0.15	75	0.05	150
F-Cyclic	0.15	75	0.05	150
F-Opportunistic	0.15	75	0.05	150
F-Balanced	0.15	75	0.05	50

Table 4.3: Parameter setting for filtering conditions. The parameters correspond to: round precision threshold (τ_r), number of queries (N), query precision threshold (τ_q), and number of documents (M).

conclusions are analogous for larger sample sizes.

Performance Metrics: We use the following metrics:

- *SampleSize@Q* and *SampleSize@D* measure the size of the document sample (i.e., number of useful documents in the sample) as a function of the number of issued queries Q and of the number of processed documents D, respectively.[7]

- *UniqueTuples@S*, *UniqueTuples@Q*, and *UniqueTuples@D* measure the quality of the sampling process in terms of the number of unique tuples and attributes as a function of sample size S, issued queries Q, and processed documents D, respectively. Specifically, we compute the number of unique tuples using case-insensitive string matching over each attribute.

- *IssuedQueries@S* and *ProcessedDocuments@S* measure the number of queries issued and documents processed, respectively, to collect a sample of size S. Given a technique and a sample size S, we only report IssuedQueries@S and ProcessedDocuments@S if the technique collects at least one sample of size S. Because not all sampling executions manage to collect document samples of all sizes[8], we complement these measures with the fraction of collections that the technique collects samples of size S from, which we define next.

[7] We do not report S or D as a percentage of the total number of documents in the collection being sampled (e.g., 50% of the documents), since we are unaware of the real size of the collection.

[8] Some sampling techniques may not reach all sample sizes S for three main reasons: (i) collections may include (very) small number of useful documents for some relations; (ii) techniques that rely on filtering conditions may terminate the sampling process early; and (iii) only a limited number of issued queries and processed documents may be allowed, for efficiency.

- *Coverage@S* measures the fraction of collections from which the sampling process manages to collect samples of size S.[9] We evaluate Coverage@S as a complementary measure to those defined above.

We run all sampling processes five times, to account for randomness, as follows: For bootstrapping-based techniques, each run uses a different initial set of seed of 20 tuples. For learning-based techniques, we built 5 disjoint training document samples from our training collection, each with 5,000 randomly picked useful documents—or the maximum number of useful documents available for each training sample[10]—and the same number of useless documents, so that the training samples are balanced. (Other seed tuples and training sample sizes yielded similar results during tuning.) Given a collection, we finally report the average over all executions using the same sampling configuration.

4.4 Experimental Results

We now report our experimental results: We start by evaluating different families of useful document retrieval techniques (Section 4.4.1). We then evaluate different query execution schedules (Section 4.4.2) and document retrieval and processing strategies (Section 4.4.3). Finally, we evaluate the impact of revising the query execution order (Section 4.4.4) and of filtering underperforming queries (Section 4.4.5).

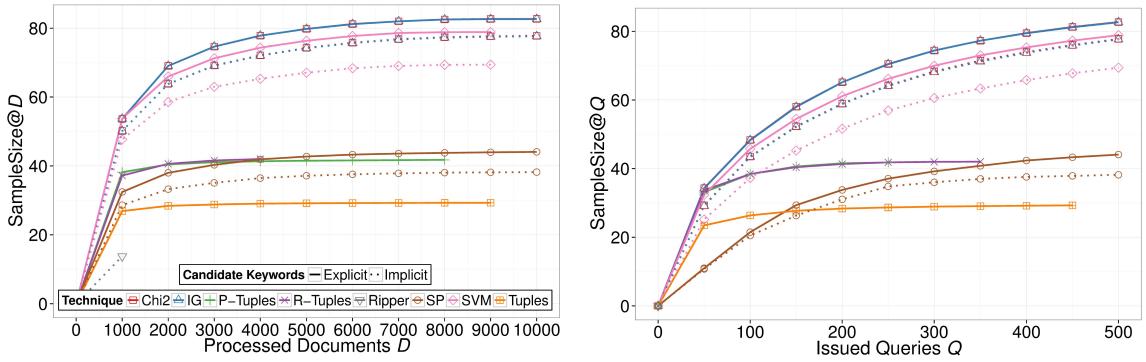
4.4.1 Impact of Useful Document Retrieval

We evaluate the two document retrieval strategies in Section 4.3, from the bootstrapping- and learning-based families discussed in Section 4.1.

Efficiency Analysis: We first evaluate efficiency by considering sample size: Figure 4.5 shows SampleSize@D (Figure 4.5a) and SampleSize@Q (Figure 4.5b) for different document retrieval strategies and processing the top-50 documents per query, for the Person–Career

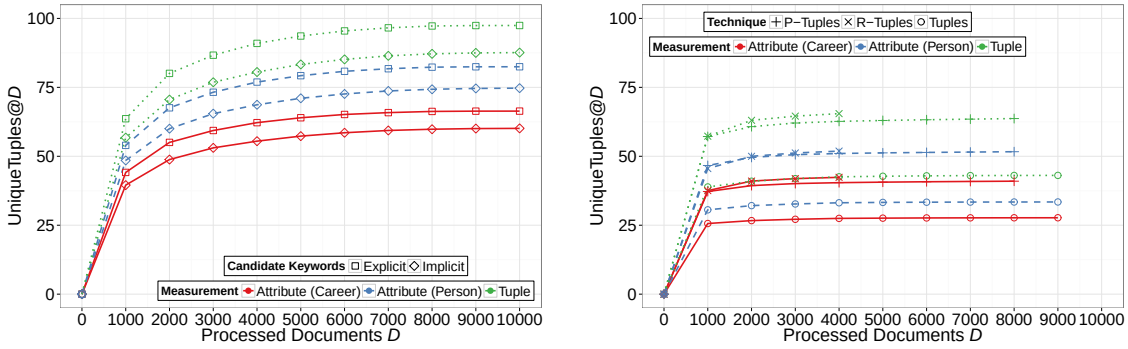
[9] We do not report Coverage@S as a fraction of the ideal coverage, since we are unaware of the real contents of the collections.

[10] For example, for the Election–Winner relation and using the SSK extraction system, each training document sample included 499 useful documents, because there were 2494 useful documents in the training collection.



(a) SampleSize@D (b) SampleSize@Q

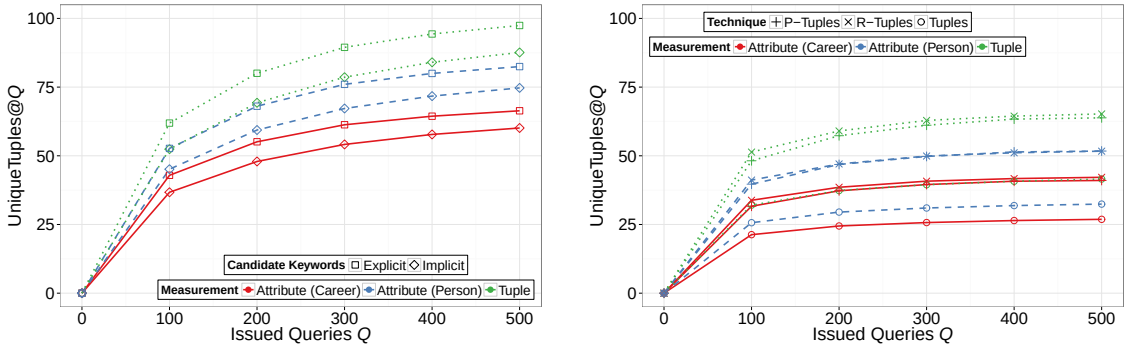
Figure 4.5: Sample size for different useful document retrieval strategies, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)



(a) χ^2 (b) All bootstrapping-based variants

Figure 4.6: UniqueTuples@D for different useful document retrieval strategies, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)

relation. (Other relations as well as number of documents per query yielded analogous conclusions.) As shown, learning-based techniques that employ keyword selection, namely, SVM, IG, and χ^2, consistently outperform other techniques after processing 1000 documents and issuing 100 queries. These techniques sample on average 100% more documents than other techniques for the same document processing and querying costs. For lower costs, bootstrapping-based techniques are comparable to keyword selection-based techniques. This finding corroborates that of previous studies for the related problem of efficiently running an extraction process over a large text collection (e.g., [AG03]), which state that bootstrapping-



(a) UniqueTuples@Q for χ^2.

(b) UniqueTuples@Q for all bootstrapping-based variants.

Figure 4.7: Number of unique tuples for different useful document retrieval strategies, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)

based techniques are rather high-precision.

The choice of candidate sets of keywords also affects sampling efficiency, as shown in Figure 4.5: The explicit candidate set of keywords, which includes values of tuple attributes in the learned queries (see Section 4.3), targets useful documents more effectively than its implicit counterpart. (We observed analogous conclusions for all relations, with the exception of Natural Disaster–Location, for which rather generic words, such as "destroyed" and "emergency", are effective and are now highly scored.) We study this relation in detail later.) This result differs from that in [AG03], where the implicit set of keywords (almost) always performed the best. We observe the largest performance gap for SVM, which gave considerably high weights to infrequent—yet discriminative—keywords in the training documents. These keywords were in turn also infrequent in our test text collections. This finding corresponds with that of previous studies (e.g., see [CLTW10]) that conclude that SVMs are many times unable to generalize to other datasets.

Quality Analysis: To evaluate the quality of the samples produced with different document retrieval strategies, we measured the number of unique tuples. Figure 4.7 shows UniqueTuples@D (Figure 4.6) and UniqueTuples@Q (Figure 4.7), processing top-50 documents per query for the Person–Career relation. For clarity, we show one learning-based strategy (see side (a) in these figures) and the bootstrapping-based techniques (see side (b)

in these figures). (Other learning-based techniques, relations, and number of documents per query yielded similar conclusions.) Our first observation is that the most efficient techniques also exhibit the highest quality: For the same document processing and querying cost, these (efficient) techniques collect a larger number of tuples, which in effect include a higher number of unique tuples. For bootstrapping-based techniques, in particular, the quality positively correlates with the domain of attributes (e.g., names of people, careers) used as queries. In the Person–Career relation, for instance, there are more people names than careers; as a result, we observe the highest quality for R–Tuples, which derives queries from the Person attribute. Unfortunately, the quality of bootstrapping-based techniques is low compared with that of χ^2 and other learning-based techniques. Moreover, these techniques reach their highest quality values early in the sampling process, which exhibits their quality limitations. This corroborates the finding in [AIG03], which states that bootstrapping-based approaches often only reach limited groups of documents—hence limited sampling quality—in the collections.

Coverage Analysis: We finally evaluate Coverage@S of the document retrieval strategies: Figure 4.8 shows Coverage@S for the learning- and bootstrapping-based variants of interest. As shown, learning-based techniques using the explicit set of keywords exhibit the highest coverage across different sample sizes. Specifically, learning-based techniques manage to collect useful documents from 30% more collections than other techniques on average. For bootstrapping-based techniques, P-Tuples collects small samples (75 documents or fewer for the Person–Career relation) from 10% and 20% more collections than R-Tuples and Tuples, respectively. For larger samples (100 documents or more, for the Person–Career relation), in contrast, R-Tuples manages to effectively collect samples from 25% and 40% more collections than P-Tuples and Tuples, respectively.

Conclusion: Based on the evaluation above, we conclude that learning-based techniques with keyword selection strategies perform the best for document sampling: They (i) collect useful documents efficiently (e.g., in terms of processed documents and issued queries); (ii) sample representative, high-quality documents for all attributes in the extraction task at hand; and (iii) manage to collect useful documents from more collections than those of other techniques.

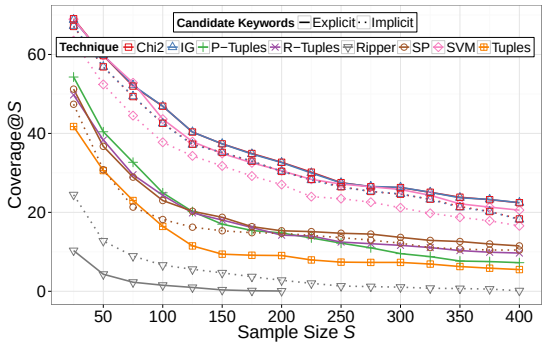


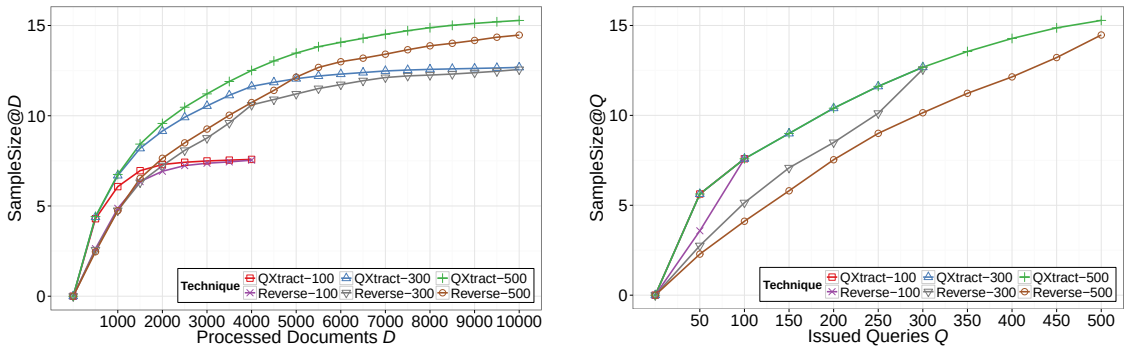
Figure 4.8: Coverage@S for different useful document retrieval strategies for different sample sizes, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.).

4.4.2 Impact of Query Execution Order

In Section 4.2, we argued that different query execution orders have distinct implications in sampling efficiency and quality. We now evaluate the discussed query execution orders: We compare (i) QXtract (see Section 4.2), to assess the performance of prioritizing effective queries; and (ii) Reverse (see Section 4.3), to assess the performance of prioritizing less-effective queries. We report our evaluation using χ^2 as query generation method and over the explicit candidate set of keywords, as it performed substantially better than other techniques and comparably to IG and SVM. We vary the number of (top) learned queries between 100 and 500.

Efficiency Analysis: To assess the efficiency of different query execution orders, we evaluated QXtract and Reverse over all relations, and for different numbers of queries: Figure 4.9 shows SampleSize@D (Figure 4.9a) and SampleSize@Q (Figure 4.9b) for different query execution orders and number of learned queries, for the Man Made Disaster–Location relation. (Other relations yielded analogous conclusions.) As shown, all versions of QXtract perform comparably or better than the Reverse counterparts: For small number of highly-effective queries (see QXtract-100 and Reverse-100 in Figure 4.9), the query execution order has almost no impact on sampling efficiency. For large numbers of queries (see QXtract-500 and Reverse-500 in Figure 4.9), the impact of the query execution order is considerable, with QXtract-500 collecting 100% more useful documents than Reverse-500.

Quality Analysis: Beyond efficiency, we also expect the query execution order to impact



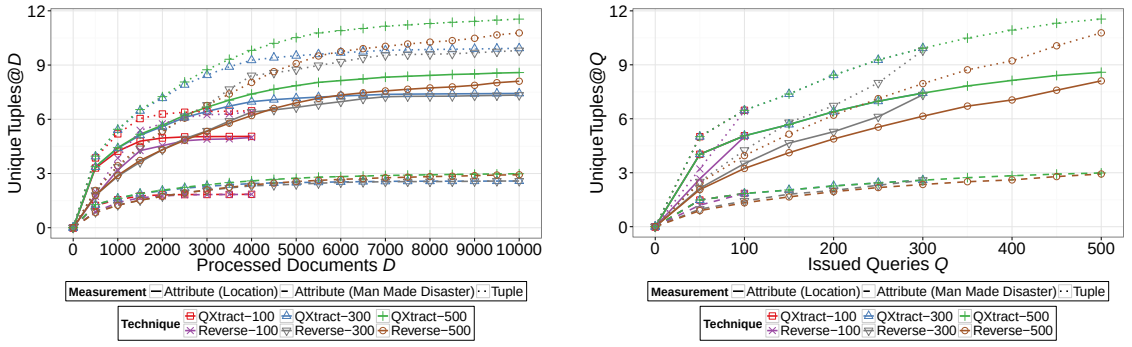
(a) SampleSize@D (b) SampleSize@Q

Figure 4.9: Sample size for different query execution orders and number of learned queries, processing 100 documents per query and for the Man Made Disaster–Location relation.

sampling quality. Figure 4.10 shows UniqueTuples@D (Figure 4.10a) and UniqueTuples@Q (Figure 4.10b), for different query execution orders and number of learned queries, and using the explicit candidate set of keywords over the Man Made Disaster–Location relation. (Other relations yielded analogous conclusions.) As shown, for the number of processed documents and issued queries, QXtract variants, which prioritize effective queries, collect a higher number of unique tuples and attributes. This happens because, as discussed above, effective queries lead to extracting more tuples—hence more unique tuples. However, we are also interested in the sampling quality of different query execution orders as a function of the sample size. This cannot be evaluated with UniqueTuples@Q and UniqueTuples@D, since we have different sample sizes across collections for the same values of Q and D.

To evaluate the intrinsic quality of different query execution orders, and to complement the quality analysis above, we evaluate sample quality across sample sizes. Figure 4.11 shows UniqueTuples@S for different query execution orders, using the explicit candidate set of keywords, and over the Man Made Disaster–Location relation. As shown, for small sample sizes (100 sampled documents or fewer), Reverse variants exhibit sample quality at least as good as that of their QXtract counterparts. This also holds for sample sizes for which QXtract has collected more samples (see Sample Size=75 in Figure 4.12).

Coverage Analysis: We finally evaluate the coverage that different query execution orders exhibit. Figure 4.12 shows Coverage@S for different query execution orders, using the Man Made Disaster–Location relation. (Other relations yielded analogous results.) Conclusions



(a) UniqueTuples@D (b) UniqueTuples@Q

Figure 4.10: Number of unique tuples for different query execution orders and number of learned queries, processing 100 documents per query and using the explicit candidate set of keywords and for the Man Made Disaster–Location relation.

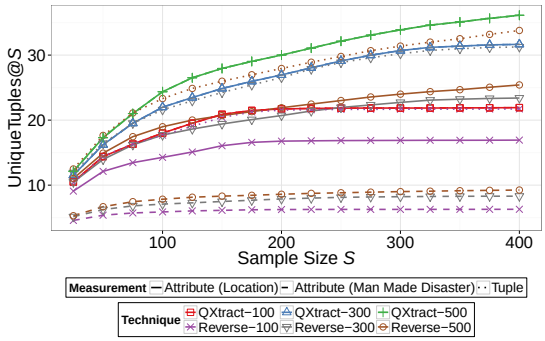


Figure 4.11: UniqueTuples@S for different query execution orders and number of learned queries, processing 100 documents per query and using the explicit candidate set of keywords and for the Man Made Disaster–Location relation.

are manifold: Focusing on a small set of highly-effective queries drastically reduces the coverage of the techniques for all sample sizes (see QXtract-100 and Reverse-100 in Figure 4.12). More importantly, the query execution order does not affect the (poor) coverage in this case. Unlike what we expected, increasing the number of learned queries showed limited impact in coverage, while its querying overhead was considerable (see Figure 4.9b).

Conclusion: We have empirically corroborated the efficiency and quality implications of different query execution orders: Prioritizing effective queries leads to more efficient sampling executions that, in turn, collect document samples from a larger number of collections than prioritizing less-effective queries. Prioritizing less-effective queries, in contrast, leads

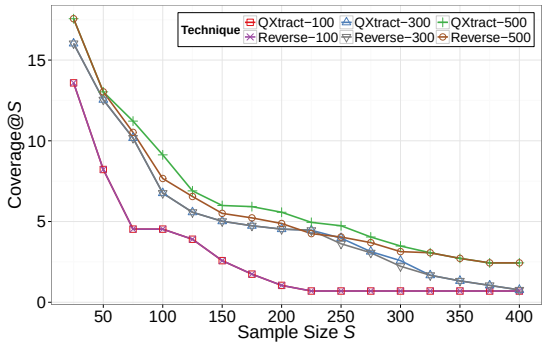


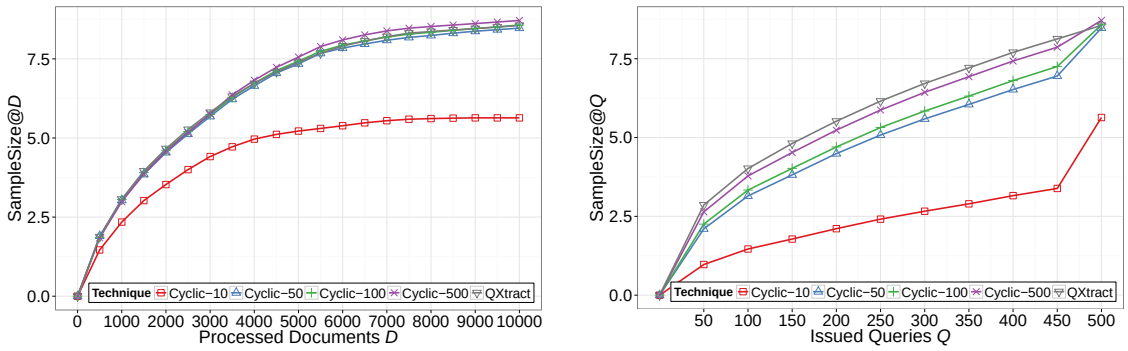
Figure 4.12: Coverage@S for different query execution orders and number of learned queries for different sample sizes, processing 100 documents per query and for the Man Made Disaster–Location relation.

to high-quality document samples, but at a considerably high document processing and querying cost. Moreover, increasing the number of learned queries has limited impact.

4.4.3 Impact of Document Retrieval and Processing

In addition to the query execution orders analyzed above, we also argued in Section 4.2 that different document retrieval and processing strategies also impact sampling efficiency and quality. We now compare: (i) QXtract, which retrieves and process documents exhaustively for each query; and (ii) Cyclic, which does so incrementally and in rounds. We report our evaluation using χ^2 as our query generation method and over the explicit candidate set of keywords, as done in Section 4.4.2.

Efficiency Analysis: We evaluate the efficiency of QXtract and Cyclic with different numbers of documents per round. Figure 4.13 shows SampleSize@D (Figure 4.13a) and SampleSize@Q (Figure 4.13b) for different document retrieval and processing strategies, and using the Person–Charge relation. (Other relations yielded similar conclusions.) As shown, there is a positive correlation between the number of documents per round and the number of sampled useful documents: QXtract and Cyclic start with highly-effective queries, which are likely to retrieve useful documents with high precision and recall. This is better illustrated in Figure 4.13b, where QXtract consistently outperforms all variants of Cyclic. In terms of processed documents, though, the sampling process benefits from moving earlier to other queries (see Figure 4.13a), because top queries may not be equally



(a) SampleSize@D (b) SampleSize@Q

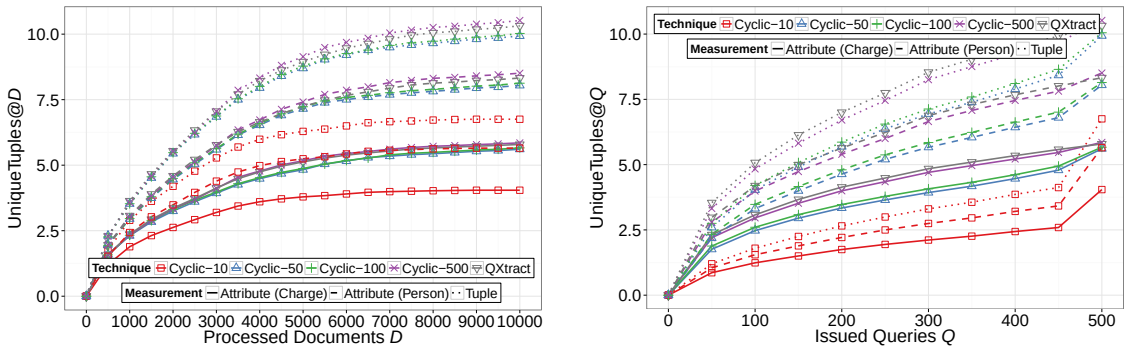
Figure 4.13: Sample size for different document retrieval and processing strategies for the Person–Charge relation.

effective across collections. As a result, variants of Cyclic with rounds of 100 documents or more collect larger samples than QXtract, for the same number of processed documents.

Quality Analysis: Beyond efficiency, we also compared the quality of different document retrieval and processing strategies. Figure 4.14 shows UniqueTuples@D (Figure 4.14a) and UniqueTuples@Q (Figure 4.14b) for different document retrieval and processing strategies, using the explicit candidate set of keywords and for the Person–Charge relation. (Other relations yielded analogous conclusions.) Surprisingly, low values of k (e.g., $k = 10$) did not enhance sample quality: Even after processing 8000 documents with Cyclic-10, sampling quality was lower than that of other variants for only 4000 retrieved and processed documents. Conversely, and similarly to what we observed for document retrieval strategies (Section 4.4.1), the number of sampled documents correlates with quality.

Coverage Analysis: Figure 4.15 shows Coverage@S for different document retrieval and processing strategies, for the Person–Charge relation. (Other relations yielded similar conclusions.) As shown, the most efficient techniques, namely, QXtract and variants of Cyclic with 100 or more documents per round, also exhibit the best coverage. Processing fewer documents per round tended to deploy querying and document processing effort on less–effective queries and useless documents, thus compromising the overall sampling performance (see Cyclic-10 in Figure 4.15).

Conclusion: Based on the evaluation above, techniques that focus on effective queries, namely, QXtract and variants of Cyclic with 100 or more documents per round, outper-



(a) UniqueTuples@D (b) UniqueTuples@Q

Figure 4.14: Number of unique tuples for different document retrieval and processing strategies, using the explicit candidate set of keywords and for the Person–Charge relation.

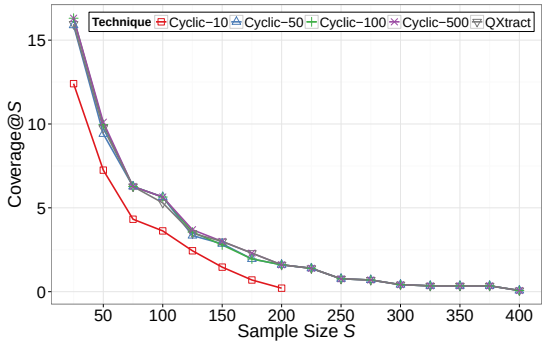


Figure 4.15: Coverage@S for different document retrieval and processing strategies for the Person–Charge relation.

formed other configurations. In particular, although these techniques perform comparably, QXtract is a better choice when querying cost dominates the sampling cost, while Cyclic prevails when document processing cost dominates sampling cost.

4.4.4 Impact of Revising Query Order

So far, our experimental evaluation is on the intrinsic performance of different query execution and document processing and retrieval strategies. However, as argued in Section 4.2, there is valuable information (e.g., the real, observed effectiveness of queries) that we can exploit along the sampling process. We now study the impact of using this information to revise the query execution order. We compare (i) Balanced and Opportunistic, which revise

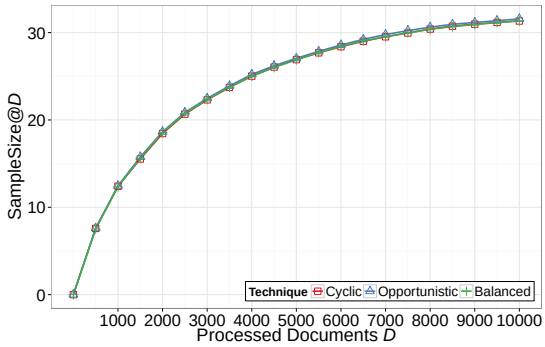


Figure 4.16: SampleSize@D for different query execution schedules and processing 50 documents per round for the Natural Disaster–Location relation.

the order of the queries; and (ii) Cyclic, which maintains their original (learned) order along the sampling process. We report our evaluation using χ^2 as our query generation method and over the implicit candidate set of keywords. Unlike in previous experiments, though, we only report the number of processed documents, as these techniques issue the same queries.

Efficiency Analysis: We first evaluate the impact on sampling efficiency of revising the query order. Figure 4.16 shows SampleSize@D for Cyclic, Opportunistic, and Balanced, processing 50 documents per round (i.e., $k = 50$) for the Natural Disaster–Location relation. (Other relations and values of k yielded analogous conclusions.) From the techniques we evaluated, Opportunistic revises the query order to prioritize queries based on their real, observed effectiveness. As expected, Opportunistic exhibits the best sampling efficiency on average. Importantly, the improvement of Opportunistic over other techniques was more noticeable over collections with a large number of useful documents.

Quality Analysis: We also evaluated the impact in sampling quality, because Balanced prioritizes less-effective queries. Figure 4.17 shows UniqueTuples@D for Cyclic, Opportunistic, and Balanced, processing 50 documents per round (i.e., $k = 50$), using the implicit candidate set of keywords, and for the Natural Disaster–Location relation. (Other relations and values of k yielded similar conclusions.) As expected, Balanced exhibits the best sampling quality for all attributes, even when Opportunistic collected more useful documents (see efficiency analysis above). More importantly, and similarly to what we pointed out above, the impact on quality of Balanced is generally more noticeable over collections that include large numbers of useful documents. These collections tend to return many

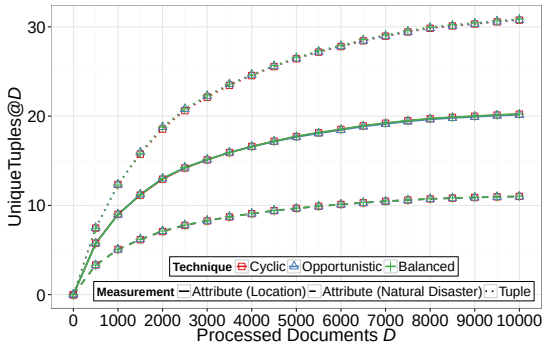


Figure 4.17: UniqueTuples@D for different query execution schedules, processing 50 documents per round, using the implicit candidate set of keywords and for the Natural Disaster–Location relation.

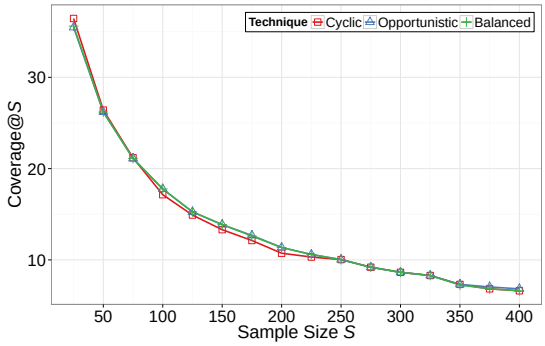


Figure 4.18: Coverage@S for different query execution schedules, processing 50 documents per round and for the Natural Disaster–Location relation.

useful documents also for less-effective queries; therefore, these queries effectively enhance sampling quality when prioritized.

Coverage Analysis: Finally, we evaluate the impact on coverage of revising query execution order. Figure 4.18 shows Coverage@S for different sample sizes for Cyclic, Opportunistic, and Balanced, processing 50 documents per round (e.g., $k = 50$) and for the Natural Disaster–Location relation. All compared techniques exhibit similar coverage, which shows that prioritizing less-effective queries based on their real, observed performance (e.g., in Balanced) does not impact the fraction of collections from which we can collect samples of different sizes.

Conclusion: Based on the evaluation above, we corroborated that we can further improve sampling efficiency and quality by accounting for the real, observed effectiveness of

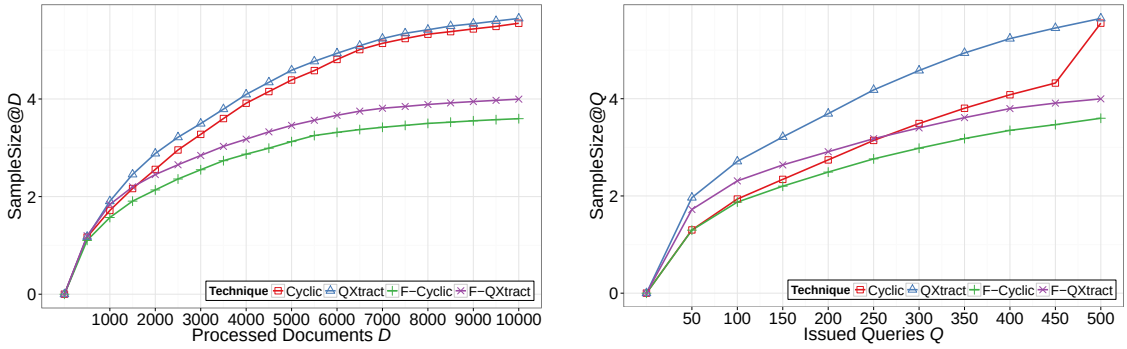
the queries. Although all techniques performed similarly, on average, Opportunistic and Balanced exhibited, respectively, the best sampling efficiency and quality, with noticeable effects on collections with large numbers of useful documents.

4.4.5 Impact of Filtering Underperforming Queries

Our last experiment involves assessing the impact of filtering underperforming queries, which, as discussed in Section 4.2, can improve sampling efficiency. We compare (i) Cyclic and QXtract, which issue and process all queries; and (ii) F–Cyclic and F–QXtract, their filtered counterparts, which filter underperforming queries using the settings of Section 4.3. Conclusions were analogous for different techniques. We report our evaluation using χ^2 as our query generation method and over the explicit candidate set of keywords.

Efficiency Analysis: We first evaluate ProcessedDocuments@S and IssuedQueries@S for different sample sizes, which we show in Tables 4.4 and 4.5, respectively. (We later analyze the coverage of these techniques, which explains why, for instance, samples of 100 documents for Cyclic are on average less expensive to obtain than those of 50 documents.) As shown, filtered versions collect samples more efficiently than their unfiltered counterparts. For example, F–Cyclic needs to process 35% fewer documents and issues 55% fewer queries than Cyclic to collect samples of 50 useful documents. The main benefit of these filtered versions is that they stop processing collections that include none—or insufficiently many—useful documents, which are a large portion of the collections. Overall, F–QXtract exhibits the best sampling efficiency across different sampling sizes; however, as we will see next, filtering underperforming queries has undesirable effects on all other relevant aspects of the sampling process.

In addition to the evaluation above, we study the impact of filtering underperforming queries on the sample size that we collect at different sampling costs. Figure 4.19 shows SampleSize@D (Figure 4.19a) and SampleSize@Q (Figure 4.19b) for both the filtered and unfiltered versions of Cyclic, processing 50 document per round (i.e., $k = 50$), and QXtract, for the Election–Winner relation. (Other relations yielded similar conclusions.) As shown, filtered versions collect on average smaller sample sizes for the same cost, because they (mistakenly) stop processing queries that would retrieve useful documents otherwise: QXtract



(a) SampleSize@D (b) SampleSize@Q

Figure 4.19: Sample size for filtered and unfiltered versions of Cyclic (using $k = 50$) and QXtract for the Election–Winner relation.

Technique	Sample Size				
	25	**50**	**100**	**200**	**400**
F–Cyclic	1067.4 ± 261	1853.2 ± 53.4	3385.9 ± 517.6	**5146.8 ± 1006.1**	-
Cyclic	2374 ± 336.2	2804.5 ± 328.6	3517.9 ± 463.1	5457 ± 567.3	**7126 ± 0**
F–QXtract	**975.1 ± 245.4**	**1761.5 ± 62.8**	**3266.4 ± 81.9**	5193.5 ± 983.2	-
QXtract	1977.3 ± 134.4	2617.4 ± 466.4	3281.4 ± 838.7	5617.9 ± 776.3	7169.5 ± 0

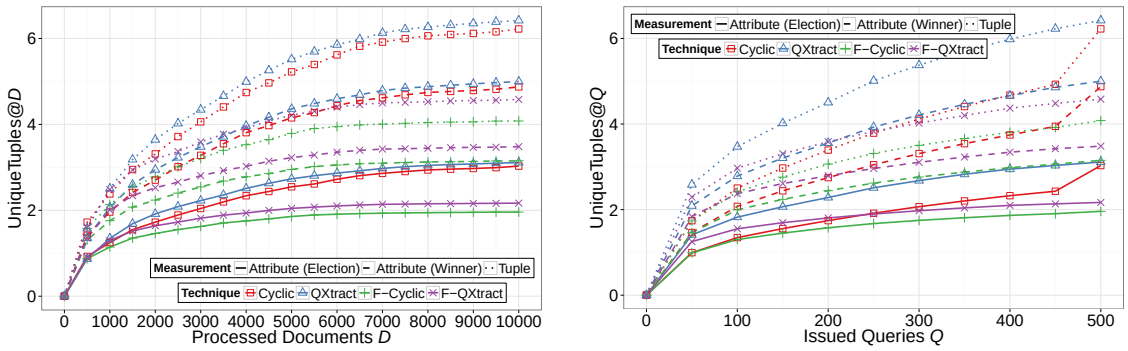
Table 4.4: ProcessedDocuments@S for filtered and unfiltered versions of QXtract and Cyclic (using $k = 50$), using the explicit candidate set of keywords and for the Election–Winner relation.

Technique	Sample Size				
	25	**50**	**100**	**200**	**400**
F–Cyclic	**83.1 ± 12.9**	128.9 ± 11.4	224.8 ± 24.8	306.5 ± 74.8	-
Cyclic	245.8 ± 19.2	316.4 ± 19.8	292.5 ± 48	374.7 ± 21.6	500 ± 0
F–QXtract	89.9 ± 23.9	**125.7 ± 15.4**	214.9 ± 11.3	305.6 ± 70.1	-
QXtract	119.9 ± 10.4	177 ± 22.3	**201.1 ± 56.5**	**298.5 ± 28.2**	**485 ± 0**

Table 4.5: IssuedQueries@S for filtered and unfiltered versions of QXtract and Cyclic (using $k = 50$), using the explicit candidate set of keywords and for the Election–Winner relation.

and Cyclic collect samples on average 100% larger than those of F–Cyclic and F–QXtract, respectively, for the same number of processed documents and issued queries.

Quality Analysis: In Section 4.2, we argued that filtering certain queries has implications for sampling quality, because the sampling process only focuses on highly-effective queries. To evaluate their real impact, we compared Cyclic, processing 50 documents per query, and QXtract to their filtered counterparts in terms of sampling quality: Figure 4.20 shows UniqueTuples@D (Figure 4.20a) and UniqueTuples@Q (Figure 4.20b), for Cyclic, QXtract, F–Cyclic, and F–QXtract, using the explicit candidate set of keywords and for the Election–Winner relation. (Other relations yielded analogous conclusions.) As expected, filtering



(a) UniqueTuples@D (b) UniqueTuples@Q

Figure 4.20: Number of unique tuples for filtered and unfiltered versions of Cyclic (using $k = 50$) and QXtract, using the explicit candidate set of keywords and for the Election–Winner relation.

underperforming queries impacts sampling quality, because less-effective queries that potentially retrieve different groups of documents may not be processed. Also, and similarly to what we observed above, the techniques that collected more useful documents for the same document processing and query issuing cost also exhibit the best sample quality, for all tuple attributes.

Coverage Analysis: We finally evaluate how filtering underperforming queries impacts the coverage of the sampling techniques. Figure 4.21 shows Coverage@S for different sample sizes for Cyclic, QXtract, F–Cyclic, and F–Qxtract, and for the Election–Winner relation. (Other relations yielded similar conclusions.) We identify two regions in this figure worth analyzing. For small samples (e.g., 200 useful documents or fewer), QXtract and Cyclic consistently cover more collections than their filtered counterparts: Filtered technique rarely reach less-effective queries. For large samples (e.g., 200 documents or more), filtered and unfiltered techniques perform similarly: Filtering conditions do not affect the (typically) few collections that include large numbers of useful documents; instead, they effectively stop processing underperforming queries and focus on the rest.

Conclusion: Based on the evaluation above, we corroborate that filtering conditions help improve the efficiency of the sampling process, but affect other relevant aspects of the sampling process. We observed that the impact of the filtering step depends on the number of useful documents in the collections: Filtered techniques are as effective as their unfiltered

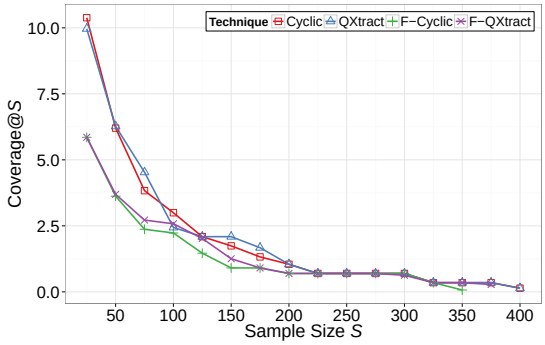


Figure 4.21: Coverage@S for filtered and unfiltered versions of Cyclic (using $k = 50$) and QXtract for the Election–Winner relation.

counterparts over collections with large numbers of useful documents, while they tend to affect collections with only a small number of useful documents considerably. As we will see in Chapter 5, we many times need to focus on collection with large numbers of documents, so filtering underperforming queries may be beneficial.

4.5 Conclusions

In this chapter, we systematically studied the problem of query-based sample generation for information extraction over a text collection. We considered (i) alternative query execution schedules, which vary on how they account for the query effectiveness, and (ii) alternative document retrieval and processing schedules, which vary on how they deploy the extraction effort over documents. Our large-scale evaluation, the first to the best of our knowledge, yielded several important conclusions: (i) schedules that focus on effective queries improve sampling efficiency, while schedules that prioritize less-effective queries favor quality; and (ii) processing the documents of highly-effective queries exhaustively consistently exhibits high sampling efficiency, but processing documents incrementally and in rounds can many times (e.g., with round sizes of 100 documents or more) exhibit better sampling efficiency and quality.

We also evaluated several different useful document retrieval methods: Learned keyword queries performed substantially better than queries derived from tuples, which have been widely used in the existing literature. Additionally, we evaluated the implications of revising

the order of the queries and of filtering underperforming queries: Revising query order during sampling helps improve sampling efficiency—when effective queries are prioritized in each round—and quality—when less-effective queries are prioritized instead. Moreover, filtering underperforming queries improves sampling efficiency considerably, although it compromises all other relevant aspects of the sampling process.

Putting it all together, the key contribution of this chapter is the development and thorough evaluation of sampling configurations that produce better-quality document samples for information extraction, and with executions that are several times more efficient, than those possible with the sampling techniques adopted in the literature. As we will see throughout this dissertation, the sampling techniques in this section were crucial for the deployment of our approaches at scale. Furthermore, the (high) quality of our document samples improves the overall performance of the extraction process. In conclusion, our results provide a roadmap for addressing this critically important building block for efficient, scalable information extraction.

Chapter 5

Ranking Text Collections for Scalable Information Extraction

In Chapter 2, we argued that information extraction is generally a computationally expensive process, and that improving its efficiency, so that it scales to large volumes of text, is of critical importance. We now introduce and address the problem of ranking text collections for an information extraction task, to prioritize the extraction effort by focusing on collections with substantial numbers of useful documents for the information extraction task. An approach for this task should rightfully conclude, for example, that FEMA [FEM15], a key up-to-date resource for natural disasters and other hazards in the United States introduced in Section 2.2.2, is better for extracting *Occurs-in* tuples than PubMed [Pub15], a well-known resource for life sciences and biomedical research with over 22 million abstracts and references to research papers also introduced Section 2.2.2. This collection ranking problem is related to the problem of resource selection in distributed information retrieval [SS11, Chapter 3], to identify topically relevant collections for a given user query. Unlike in distributed information retrieval, though, our information extraction scenario requires that we identify collections with useful documents for the extraction task, rather than documents that are topically relevant for a given query. Despite this difference in focus, we can adapt resource selection approaches to our information extraction scenario, as we will see, as well as develop alternative, information extraction-specific approaches.

To effectively rank text collections for a given extraction task, we develop approaches that target the useful documents for the extraction task in question. We compare both (adaptations of) state-of-the-art resource selection strategies and information extraction-specific approaches in an extensive experimental evaluation over realistic Web text collections, and for several different extraction tasks. In summary, the contributions of this chapter are:

- We review (adaptations of) traditional approaches for estimating, for each text collection, the number of useful documents for a given extraction task (Sections 5.2 and 5.3).
- We present information extraction-specific approaches for estimating, for each text collection, the number of useful documents for a given extraction task (Section 5.4).
- We report the results of an extensive evaluation of both (adaptations of) traditional approaches for distributed information retrieval, and information extraction-specific approaches over real-world Web collections and for several different information extraction tasks. Our results show the merits and limitations of the alternative families of approaches, and provide a roadmap for addressing this critically important building block for efficient, scalable information extraction (Sections 5.5 and 5.6).

We now review necessary background and introduce the problem of ranking text collections for efficient and scalable information extraction, our problem of focus in this chapter (Section 5.1). The bulk of this chapter has been published as [BGD15].

5.1 Background and Problem Definition

To run an information extraction system over the available text collections, a naïve, expensive approach could resort to state-of-the-art approaches for efficient query-based information extraction execution (e.g., [AG03; FC11; BLNP11a]) over each collection individually, as described in Section 2.3.[1] Such a naïve approach would be unnecessarily expensive,

[1]Our approach is not applicable over open information extraction scenarios (e.g., [BCS+07]) where documents frequently contribute tuples to the open-ended extraction task.

because not all collections contain any useful documents. Therefore, to prioritize the information extraction effort, for efficiency, we focus on the problem of ranking text collections for an information extraction task of interest. Our approaches should be applicable to fully-accessible text collections as well as to deep web text collections, discussed in Section 2.2.

A related problem, *resource selection*, has been studied in the context of distributed information retrieval, to rank collections according to their topical relevance to a given user query [SS11]. Resource selection approaches generally consist of two steps: (1) *descriptor generation:* in an offline step, build a compact, representative collection summary (e.g., consisting of word frequency vectors [CLC95; GGMT99] or document samples [SC04; SZ07]); (2) *relevance estimation:* to process a given query, use the collection descriptors to estimate the number of topically relevant documents in each collection, and rank the collections accordingly.

Unlike in distributed information retrieval, our information extraction scenario requires that we identify useful collections, or collections with useful documents for the extraction task, rather than collections with documents that are topically relevant to a given query. As a result, the collection descriptors will need to effectively capture the characteristics of the useful documents, a challenging proposition because of two critical reasons. First, the notion of document usefulness is, by definition, specific to a given extraction task, so our collection ranking approaches—and the collection descriptors on which they rely—will have to be flexible to adapt to each given extraction task. In particular, the "one-size-fits-all" descriptors adopted by resource selection approaches for distributed information retrieval would not be appropriate for our information extraction scenario. Second, the fraction of documents in a collection that are useful for an information extraction task can many times be very small (see Section 2.3), so our collection ranking approaches—and the estimation techniques on which they rely—will have to effectively target the useful documents, to keep the ranking overhead to manageable levels.

To prioritize text collections for an extraction task, we must identify the most useful collections, namely, the collections with the largest numbers of useful documents for the IE task. Therefore, we need to estimate the number of useful documents in each collection and, importantly, we need to do so efficiently (e.g., by issuing a relatively small number

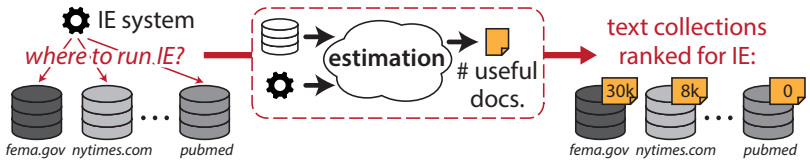


Figure 5.1: Collection ranking for information extraction.

of queries to each collection). For this estimation problem, we could exploit state-of-the-art techniques for measuring certain (queryable) collection properties (e.g., their number of documents) [ZZD11; BYG11; ZZD13]. Unfortunately, as we will see, such techniques can be prohibitively expensive for information extraction, because they may need to issue many queries to sufficiently cover the (often rare) useful documents for an extraction task of interest.

We summarize the problem that we address in this chapter as follows:

Problem Definition 2 *Consider a set of text collections and an information extraction task T with its corresponding (previously trained) information extraction system. Our goal is to rank the collections according to their number of useful documents for the IE task T (see Figure 5.1). Furthermore, the ranking process should be efficient (e.g., in terms of the number of queries issued to each collection), to keep its overhead to reasonable levels.*

Earlier efforts to identify collections for an extraction task (e.g., [JS09; AC05]) have focused on the quality of the extraction output, rather than its volume. The (complementary) methods described in this chapter can be adapted to consider quality (see Chapter 10).

5.2 Overview of Estimation Approaches

To prioritize the information extraction effort and rank text collections for an extraction task, we need to estimate the number of useful documents for the information extraction task in each collection. Specifically, for each collection C, we will estimate the cardinality of C^{U}, the set of useful documents in C for the information extraction task at hand. In this section, we provide an overview of three families of state-of-the-art estimation approaches that we can adapt for the task.

Given an information extraction task, we can cast the problem of estimating the number of useful documents for the task in a text collection as an instance of the generic task of estimating a "property" of interest for a text collection, which has been studied extensively in the research literature. Such a property F of a collection C is typically defined as an aggregate over a document-level function, say, $f(d)$. A simple example is the estimation of the number of documents in a text collection C: in this case, $f(d) = 1$ for every document d in C, and $F(C) = \sum_{d \in C} f(d)$. In our information extraction scenario, to estimate the number of useful documents we should define $f(d) = \mathbb{1}\{d \text{ is useful}\}$, that is, as the indicator function that returns 1 if d is useful and 0 otherwise. Various methods have been proposed to estimate properties of (queryable) document collections (e.g., collection size, number of documents relevant to a query, average document length) [BYG11; HSB+10; ZZD11; ZZD13], and these methods can be classified in three broad classes: (i) surrogate-based methods, (ii) query pool-based methods, and (iii) query pool-free methods.

Surrogate-based methods construct an approximate representation of the entire collection, and then use that surrogate to estimate the metric of interest, without further accessing the actual collection. A surrogate typically comprises a (relatively small) document sample (e.g., [SC04; SZ07]), or document frequency estimations for the terms occurring in the collection (e.g., [CLC95; GGMT99]). In resource selection for distributed information retrieval, such representations have been widely used to estimate the number of documents relevant to a query (e.g., CORI [CLC95], GlOSS [GGMT99], ReDDE [SC04], and Relax [SZ07]). For example, ReDDE builds a random document sample S for each collection C, once and for all. Simply stated, to judge the relevance of C to a query q, ReDDE extrapolates the number of documents relevant to q in S to the entire collection.

Query pool-based methods pick queries from a predefined query pool Q (e.g., a dictionary of words or n-grams collected from extensive web crawls) to retrieve—from the collection at hand—documents from which to estimate the metric of interest. Unlike the collection-specific surrogates, the query pool can be shared across collections and can also be targeted specifically to the estimation task at hand. The query pool-based method in [LSYM01] aims at estimating collection size effectively but is inefficient: for large collections, the samples required to produce accurate estimates are very large. Subsequent (query pool-based) meth-

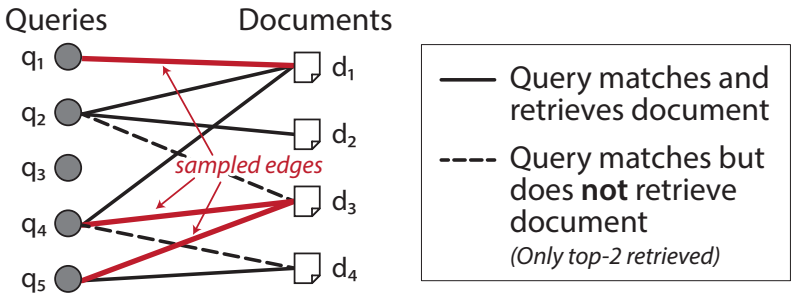


Figure 5.2: An example (query, document)-graph: the estimate contribution $f(d_1)$ from document d_1 has a $\frac{1}{3}$ weight (since its sampled degree is 3) and $f(d_3)$ is counted twice, each time with weight $\frac{1}{2}$.

ods addressed this limitation, and sample random edges from a (query, document)-graph, as sketched in Figure 5.2: the graph vertices are queries $q \in Q$ and documents $d \in C$, a solid edge (q, d) means that q retrieves d, and a dashed edge (q, d) means that q is mentioned in the contents of d but not retrieved (see Section 2.2.2 for possible reasons of this case). For each sampled (q, d) pair, the measure $f(d)$ contributes to the estimation of $F(C)$ with a weight that is proportional to the probability of having sampled d. Pool-based methods work well, provided the query pool Q retrieves all documents of interest for the metric (i.e., for which $f(d) \neq 0$).

Query pool-free methods avoid relying on a query pool, and rather find the queries to issue "on the fly." These methods use a seed query (e.g., a common word or phrase) to retrieve a first set of documents and select the next query to issue from these documents. Thus, they issue queries and retrieve documents to perform a random walk on a graph where nodes are either queries [ZZD13] or documents [BYG08]. To properly weigh the particular $f(d)$ value derived from visiting a node in such graph, these methods use the fact that the probability of visiting a node during a random walk is proportional to its number of incident edges.

In our information extraction scenario, we need to estimate the number of useful documents in a text collection for an information extraction task of interest. Unfortunately, only very few or no useful documents might be present in a truly random document sample from the collection, given that useful documents might be rare for an information extraction task. This poses a problem for all three families of estimation methods summarized above.

<table>
<tr><th>Family</th><th>Technique (B or IE)</th><th>Collection</th><th colspan="2">IE</th></tr>
<tr><td rowspan="2">Surrogate</td><td>ReDDE (B) (Section 5.3.1)</td><td>Document sample + size estimate</td><td colspan="2">Queries</td></tr>
<tr><td>Surrogate (IE) (Section 5.4.1)</td><td>Term-frequency map + size estimate</td><td colspan="2">Document sample</td></tr>
<tr><td rowspan="4">Query Pool-Based</td><td rowspan="2">PB (B) (Section 5.3.2)</td><td>In sum: -</td><td rowspan="2">In generic: -</td><td rowspan="2">In specific: query pool</td></tr>
<tr><td>In avg: size estimate</td></tr>
<tr><td rowspan="2">PB-W (IE) (Section 5.4.2)</td><td>In sum: -</td><td rowspan="2">In generic: weights</td><td rowspan="2">In specific: query pool + weights</td></tr>
<tr><td>In avg: size estimate</td></tr>
<tr><td rowspan="2">Query Pool-Free</td><td>PF (B) (Section 5.3.3)</td><td>Query sample</td><td colspan="2">-</td></tr>
<tr><td>PF-W (IE) (Section 5.4.3)</td><td>Query sample</td><td colspan="2">Query sample</td></tr>
</table>

Table 5.1: Summary of the characteristics of the baseline (B) and information extraction-specific (IE) methods in Chapter 5.

The next sections describe the existing approaches in detail (Sections 5.3.1 through 5.3.3) and derive information extraction-specific estimators that effectively target useful documents (Sections 5.4.1 through 5.4.3). By aiming to collect documents for which $f(d) \neq 0$, our information extraction-specific methods are designed to have more non-zero terms in their estimation, thus alleviating the limitations of existing estimation techniques for other tasks. Table 5.1 summarizes the requirements of each estimation method to handle collections and information extraction tasks.

5.3 Traditional Estimation Approaches: Adaptation for Collection Usefulness

We now review relevant details of the traditional estimation approaches that we consider in this chapter, and adapt them to our collection usefulness estimation. Specifically, we consider surrogate-based (Section 5.3.1), query pool-based (Section 5.3.2), and query pool-free (Section 5.3.3) estimators.

5.3.1 Surrogate-Based Estimator

Our first (baseline) estimator is an adaptation of ReDDE, a resource selection technique for distributed information retrieval [SC03]. To estimate the number of topically relevant documents for a query q in a collection C, ReDDE predicts the relevance of a representative sample $S \subset C$ and scales it to the entire collection with a factor $SF = |C|/|S|$ to obtain a collection relevance metric $\text{Rel}(q, C)$. This metric relies on estimating (i) the collection size $|C|$ and (ii) the relevance of sample documents in S to the given query q. The size $|C|$ is query-independent and thus computed once and for all (e.g., using the sample-resample method [SC03]), while the query relevance for C is based on issuing q to a centralized global sample unifying all individual collection samples.

Specifically, the relevance of query q for a collection C is measured in terms of the expected number of relevant documents $\widehat{Rel_q^C}$ in C for q, defined as:

$$\widehat{Rel_q^C} = \sum_{d_i \in S} P(\text{rel}|d_i, q) \cdot SF,$$

where $P(\text{rel}|d_i, q)$ is the probability of document d_i being relevant to q. Here, $P(\text{rel}|d_i, q)$ is computed using an estimated global document ranking obtained from issuing q to a global collection (that includes all documents from all collections): ReDDE will assign a query-specific constant C_q to the top-n documents (e.g., 1% of the total number of documents) and 0 to all other documents. The position $\sigma(d_i)$ of document d_i in the global ranking is computed as:

$$\sigma(d_i) = \sum_{\sigma_S(d_j) < \sigma_S(d_i)} \frac{|C^{d_j}|}{|S^{d_j}|},$$

where $\sigma_S(d_i)$ is the position of document d_i in the (ranked) list of documents retrieved by q from the centralized sample, for a given effective retrieval method (e.g., INQUERY [CCB95]), and C^{d_j} and S^{d_j} are the collection of document d_j and its document sample. Intuitively, $\sigma(d_i)$ estimates the number of documents that would be ranked higher than d_i in a global ranking for query q.

To apply ReDDE to our information extraction scenario, we could intuitively replace the relevance of a document with the real usefulness of d (i.e., $f(d) = \mathbb{1}\{x \text{ is useful}\}$),

obtained by running the information extraction system at hand over d. However, this is problematic: Finding a single query that effectively models the information extraction task at hand is complicated, given the variations in language to express extraction tasks and that standard keyword search identifies documents whose topic is relevant to queries, without considering their relevance to the extraction task at hand. More importantly, a random sample might have very few or no useful documents because, as discussed, useful documents for an information extraction task are often rare. This may lead to many zero estimates.

Instead, we model the information extraction task as a set of high-performing queries Q_{IE}, which we can automatically learn (e.g., see QXtract [AG03]). We thus calculate collection relevance for the information extraction task as a sum of the ReDDE relevance metric, taken over the top-k queries from our set of information extraction-specific queries:

$$|\hat{C^{\text{U}}}| = \sum_{q \in \text{top}_k(Q_{\text{IE}})} \text{Rel}(q, C).$$

Importantly, and in contrast to the original ReDDE proposition, in our (multiple-queries) formulation documents can potentially be retrieved by many queries. To avoid computing the relevance of documents multiple times, we account for their relevance once and for all. Furthermore, $|\hat{C^{\text{U}}}|$ is not an absolute estimate of the number of useful documents, because the few queries in $\text{top}_k(Q_{\text{IE}})$ might be far from comprehensively expressing all variations of mentions for the extraction task at hand. Rather, $|\hat{C^{\text{U}}}|$ is a relative estimate, where higher values indicate higher usefulness.

5.3.2 Query Pool-Based Estimator

We consider the method of Bar-Yossef et al. [BYG11], which set the foundations for subsequent query pool-based estimators (e.g., [ZZD11]) and allows estimating any generic metric that can be expressed as a discrete integral over the documents in C:

$$\text{Int}_\pi(f) \triangleq \sum_{d \in C} f(d) \cdot \pi_D(d),$$

where f is the target function of interest and π_D weighs documents as needed (e.g., $\pi_D(d) = 1$ if all documents contribute equally). As indicated in Section 5.2, documents $d \in C$ are obtained by issuing queries from a pool Q. The method of Bar-Yossef et al. relies on two core ideas: (i) extend π_D to a measure over the (query, document)-space $Q \times C$, and (ii) apply importance sampling to use a practical sampling strategy, selecting a (query, document)-pair with probability $p(q, d)$, rather than from the probability distribution induced by π_D, which is unfeasible to sample from. For Item (i), the extended measure is:

$$\pi(q, d) \triangleq \frac{\mathbb{1}\{d \in C_q\} \cdot \pi_D(d)}{\omega(d)},$$

where C_q is the set of documents that q retrieves from C and $\omega(d)$ is the degree of document d, defined as the number of queries in Q that retrieve d. With this extended measure π, referred to as the *target distribution*, and defining $g(q, d) \triangleq f(d)$, it is easy to see that $\text{Int}_\pi(g) = \text{Int}_{\pi_D}(f)$. For Item (ii), the practical sampling strategy is: (1) pick a random query q from the set of queries that return at least one document, noted as Q_+, and then (2) randomly pick one of the documents it retrieves:

$$p(q, d) \triangleq \frac{1}{|Q_+|} \cdot \frac{\mathbb{1}\{d \in C_q\}}{|C_q|}.$$

In importance sampling, $p(q, d)$ induces a probability distribution referred to as the *trial* distribution. The idea is then to sample (query,document)-pairs according to this trial distribution, and correct the estimate with a factor $w(q, d)$ to obtain the following (corrected) estimator[2]:

$$\text{IS}(q, d) = f(d) \cdot \frac{\pi(q, d)}{p(q, d)} = f(d) \cdot w(q, d). \tag{5.1}$$

Here, $w(q, d)$ is referred to as the *importance weight*, and is defined as:

$$w(q, d) \triangleq \frac{\pi(q, d)}{p(q, d)} = \frac{\pi_D(d) \cdot |Q_+| \cdot |C_q|}{\omega(d)},$$

[2]This is valid as long as $\mathsf{supp}(p) \supseteq \mathsf{supp}(\pi)$, where $\mathsf{supp}(p) \triangleq \{x \in C : p(x) > 0\}$.

which yields the unbiased importance sampling estimator for $\text{Int}_\pi(f)$ when used in (5.1). Bar-Yossef et al. use an efficient estimator $u(q,d)$ of $w(q,d)$, defined as $u(q,d) = \pi_D(d) \cdot \text{PSE} \cdot |C_q| \cdot \text{IDE}(d)$, with a pool size estimator PSE for $|Q_+|$ and an inverse degree estimator $\text{IDE}(d)$ for $1/\omega(d)$, in turn, to calculate the approximate importance sampler (AIS):

$$\text{AIS}(q,d) \triangleq f(d) \cdot u(q,d)$$

The authors show that if u approximates w well, and if the ratio u/w is uncorrelated with f, AIS remains largely unbiased.

However, directly using AIS in our information extraction scenario is problematic: (i) the number of queries to issue and subsequent information extraction-processing of returned documents to determine $f(d)$, to find some useful documents, may be high; and (ii) the estimation will have high variance because of the few non-zero $f(d)$ values.[3] To address the first limitation for the related problem of counting the frequency of a given word (e.g., "sports") in a collection, Zhang et al. identify the queries that are positively correlated with the word (e.g., query [golf]) [ZZD11]. The query sampling process is then stratified over correlated and uncorrelated queries. Unfortunately, this approach still requires issuing a large number of queries.

Estimator for Averages: A variant of the above estimator for sums [BYG11] computes:

$$\text{Avg}(f) = \frac{\sum_i f(d) \cdot u(q_i, d_i)}{\sum_i u(q_i, d_i)},$$

where the PSE factor cancels out, and obtains the estimator by multiplying $\text{Avg}(f)$ by the size of the collection. This collection size estimation can be done once and for all, and reused for different metrics (e.g., for the usefulness for different information extraction tasks), to amortize its cost. This estimator allows for a more efficient, yet low-bias estimator for $1/\omega(d)$, that derives directly from the contents of d, incurring no additional querying cost: (1) generate all possible queries from the contents of d and (2) count their incidence in Q. Such a degree estimator, however, would incur substantial bias if applied directly on an

[3] The variance can be reduced via Rao-Blackwellization, as suggested in [BYG11], which requires running information extraction over all retrieved documents. We evaluate this version of the algorithm later in the experimental section (Section 5.6).

estimator for sums [BYG11].

5.3.3 Query Pool-Free Estimator

We focus on the method introduced by Zhang et al. [ZZD13], using a query graph:[4] (i) the nodes are h-grams[5] q that retrieve at least u documents[6] and (ii) undirected edges connect a node pair (q, q') if q' matches (i.e., appears in the text of) at least one of the documents that q retrieves, and vice versa. Since a random walk implies that a node q is visited with a probability proportional to its degree $d(q)$, each per-query estimate is weighted with $1/d(q)$ to agree with uniform sampling. To estimate $d(q)$, we retrieve all documents C_q returned by issuing q, get the h-grams they contain as Q' (as potential neigbors of q) and then sample (uniformly at random, with replacement) a $q' \in Q'$, until we find such a q' that (i) retrieves at least u documents, and (ii) q is among the h-grams found in the documents $C_{q'}$. If we need n tries to find such q', then we estimate $\hat{d(q)} = |Q'|/n_{\text{ALL}}$. The estimation of a function $F(C) = \sum_{d \in C} f(d)$ from sampled queries S from graph Q is:

$$\hat{F}(C) = |\hat{Q}| \cdot \underbrace{\frac{\sum_{q \in S} 1/d(q) \cdot \sum_{d \in C_q} f(d)/\omega(d)}{\sum_{q \in S} 1/d(q)}}_{\#\text{ useful documents per sampled query}}$$
$$= |V_c| \cdot \tilde{\lambda}(V_c) \cdot \frac{\sum_{q \in S} 1/d(q) \cdot \sum_{d \in C_q} f(d)/\omega(d)}{\sum_{q \in S \cap V_C} 1/d(q)}. \quad (5.2)$$

[4]The pool-free approach in [BYG08] defines a document graph where: (i) a pair of documents (d, d') is connected if they are retrievable by the same query (i.e., they are connected to the same query q in the (query,document)-graph of Figure 5.2), and (ii) the edge between two documents (d, d') is weighted with the number of such queries they share. The limit distribution $p(d)$ of the random walk corresponds to selecting document d proportionally to its degree $\omega(d)$ in the (query, document)-graph. To rescale this distribution to a uniform target distribution $\pi(d)$, we could adopt a similar idea to that of the pool-based approaches above: We can distribute the weight of a document across its edges and obtain an unbiased estimator. Unfortunately, this document graph approach is inefficient, since it may issue a few million queries to obtain only a few thousand documents.

[5]Zhang et al. argue that $h = 1$ works well in practice.

[6]Parameter u controls the size and connectivity of the graph. Zhang et al. propose $u = 3$.

Here, $\omega(d)$ is the number of queries in Q that retrieve d. Furthermore, the size of the query graph $|\hat{Q}|$ is estimated from a startup query collection V_C.[7] Specifically, V_C is a sample of the vocabulary of h-grams appearing in the collection, obtained from all the documents of another random walk, and that is independent of the random walk to obtain S. Because many h-grams in V_C retrieve fewer than u documents, $\tilde{\lambda}(V_C)$, an unbiased estimator of the fraction of h-grams in V_C that retrieve at least u documents, is used to correctly compute $|\hat{Q}|$. In [ZZD13], $\tilde{\lambda}(V_C)$ is assessed by drawing a random sample of h-grams in V_C, issuing them to the collection at hand, and computing the fraction of them that retrieve u documents or more.

While eliminating a potential coverage issue by avoiding an a priori query pool, the resulting estimation may still require many queries in the information extraction scenario, to find sufficiently many useful documents.

5.4 Information Extraction-Specific Estimators for Collection Usefulness

In the previous section, we described traditional estimation approaches for different families of estimation approaches. We now describe our information extraction specific estimators for these relevant families. Specifically, we describe surrogate-based (Section 5.4.1), query pool-based (Section 5.4.2), and query pool-free (Section 5.4.3) estimators.

5.4.1 Targeted Surrogate-Based Estimator

An alternative, information extraction-specific estimation approach to the surrogate-base method in Section 5.3.1 could be simply to collect a random sample S from a collection C, run the information extraction system over the documents in S, and extrapolate the number of useful documents to the full collection C as in ReDDE, namely, $|\hat{C^{\text{U}}}| = SF \cdot |S^{\text{U}}|$ with scaling factor $SF = |C|/|S|$. Unfortunately, a random sample might have very few or no useful documents because, as discussed. To address this problem, the document sample S should be then biased towards useful documents.

[7] We correct an erroneous $1/|S|$ factor from [ZZD13, eq. (5)].

The proposition above implies that we subsequently have to correct the scaling factor SF for the usefulness bias of S. The main idea is to look at document frequency differences of certain terms between the sample S and the full collection C. For a given term t, let $df(t, X)$ be the fraction of documents in X that contain t, and define the frequency ratio in the sample vs. in the full collection as $\alpha_t^S \triangleq \frac{df(t,S)}{df(t,C)}$. We propose to use the average of this ratio over *useful terms*, T^{U}, as a heuristic scaling factor: $SF = \mathbb{E}_{T^{\mathrm{U}}}\left[1/\alpha_t^S\right]$, where useful terms are those that are (i) biased towards usefulness, in contrast to "neutral" terms that would appear equally in useful and useless documents; and (ii) overrepresented in the sample (i.e., $\alpha_t^S > 1$). The rationale for this choice of SF is that the overrepresentation of such useful terms in S compared to in C is a good proxy for the overrepresentation of useful documents. (We can find T^{U} through standard statistical significance tests.)

A formal motivation of our SF starts from stochastic variables defined for a document d: (i) S: the event that d is selected in the sample, (ii) U: the event that d is useful, and (iii) T: the event that d contains a given term t. We thus can write (applying Bayes' rule and $P(U|S) \cdot P(S) \cdot |C| = P(U, S) \cdot |C| = |S^{\mathrm{U}}|$ for the last two transitions):

$$|C^{\mathrm{U}}| = P(U) \cdot |C| = \frac{P(U|S) \cdot P(S)}{P(S|U)} \cdot |C| = \frac{1}{P(S|U)} \cdot |S^{\mathrm{U}}|.$$

Now, we claim that $P(S|U) \approx P(S|T)$ holds for *useful terms*: (a) if a term is highly useful, we expect it to appear in useful documents (regardless of the adopted sampling strategy), i.e., $P(T|U, S) \approx P(T|U)$ or, more specifically, $\frac{P(T|US)}{P(T|U)} \approx 1$; (b) if a term is overrepresented in the sample, then the probability of sampling a document including this term is independent on the document's usefulness and, consequently, $P(S|U, T) \approx P(S|T)$.

Combining Item (a) and Item (b) corroborates our claim. We thus corroborate our hypothesis as:

$$P(S|U) = \frac{P(S|UT) \cdot P(T|U)}{P(T|U, S)} \approx P(S|U, T) \cdot 1 \approx P(S|T)$$

Thus, for a useful term t (with $X_t \triangleq$ documents in X containing t):

$$|C^{\mathrm{U}}| = \frac{1}{P(S|U)} \cdot |S^{\mathrm{U}}| \approx \frac{1}{P(S|T)} \cdot |S^{\mathrm{U}}| = \frac{P(T)}{P(S) \cdot P(T|S)} \cdot |S^{\mathrm{U}}|$$
$$= \frac{P(T)}{P(S,T)} \cdot |S^{\mathrm{U}}| = \frac{|C_t|/|C|}{|S_t|/|C|} \cdot |S^{\mathrm{U}}| = \frac{df(t,C)}{df(t,S)} \cdot |S^{\mathrm{U}}| = \frac{1}{\alpha_t^S} \cdot |S^{\mathrm{U}}|.$$

Importantly, we assume that we know the document frequencies $df(t, C)$ in the complete collection, as well as the collection size $|C|$. We can estimate these values reliably once and for all for each collection (e.g., see [IG08] and [SC03]).

5.4.2 Targeted Query Pool-Based Estimator

For our query-pool based estimator, we adapt the Bar-Yossef et al. approach in Section 5.4.2, with target and trial distributions that are aligned with $f(d)$ for usefulness: Our *target distribution* assigns probabilities greater than 0 only to useful documents:

$$\pi_{\mathrm{U}}(q,d) \triangleq \frac{\mathbb{1}\{d \in C_q^{\mathrm{U}}\} \cdot \pi_D(d)}{\omega(d)},$$

where C_q^{U} is the set of useful documents that q retrieves from C. Our *trial distribution*, accordingly, should (i) retrieve useful documents with high recall and precision, and (ii) be efficient to sample from.

One possible approach for our trial distribution would be to define a query pool that is specific to the extraction task at hand to, in turn, directly apply state-of-the-art estimators such as those above (e.g., [BYG11; ZZD11]). For example, a query pool for our *Occurs-in* relation would include words correlated with natural disasters (e.g., "richter", "hurricane", "aftermath"). To automatically generate these queries, we could resort to the learning approach in QXtract [AG03]. Unfortunately, these queries exhibit far-from-perfect recall [BLNP11a; AG03], in that not all useful documents can always be retrieved. (i.e., the aforementioned coverage issue, which is difficult to assess and correct). Moreover, precision would also be compromised: relevance to the issued query does not necessarily imply a document would contain extractable tuples.

To alleviate the recall and precision limitations above, and to retrieve useful documents

with high recall and precision, we propose to give higher selection weight to (potentially) useful queries, rather than focusing on a few queries correlated to the extraction task. Specifically, for recall, we obtain our query pool from a large, external text collection E that we can process once and for all. For precision, we learn the usefulness of—and assign proportional selection weights to—queries with respect to the given information extraction task: (1) process E with the information extraction system at hand, to identify the useful documents (2) query E with all words in the documents, and (3) count the useful documents within the top-k results for each query q, noted as $|E_q^{\text{U}}|$. Importantly, the step Item (1) above requires that the external collection E has useful documents, which may be non-trivial for some information extraction tasks. Such assumption is acceptable, however, because the cost of finding a collection with useful documents has to be paid once and for all. Finally, our trial distribution assigns a selection weight to each query q, defined as $w \cdot |E_q^{\text{U}}| + |E_q^{\text{N}}|$, where $|E_q^{\text{N}}|$ is the number of useless documents within the top-k results for q.[8]

Given these query selection weights, our trial process consists of two steps: (1) pick a useful query q, namely, a query that retrieves at least one useful document, proportionally to its weight, and (2) pick a document from q's useful results uniformly at random. Specifically, for Item (1), we need to issue queries until retrieving (at least) one useful document. For Item (2), we need to sample documents uniformly at random from q's results and process them with the extraction system until we find a useful document. This yields the following trial distribution:

$$p_{\text{U}}(q, d) \triangleq \frac{w \cdot |E_q^{\text{U}}| + |E_q^{\text{N}}|}{\mathbb{Z}_w} \cdot \frac{\mathbb{1}\{d \in C_q^{\text{U}}\}}{|C_q^{\text{U}}|}.$$

Here, $\mathbb{Z}_w = \sum_{q \in Q_+^{\text{U}}} w \cdot |E_q^{\text{U}}| + |E_q^{\text{N}}|$ is the normalization factor of the probability distribution induced by the queries that retrieve at least one useful document, Q_+^{U}. We can now obtain our importance weight function as:

$$w_{\text{U}}(q, d) \triangleq \frac{\pi_{\text{U}}(q, d)}{p_{\text{U}}(q, d)} = \frac{\mathbb{Z}_w \cdot \pi_D(d)}{\omega(d)} \cdot \frac{|C_q^{\text{U}}|}{w \cdot |E_q^{\text{U}}| + |E_q^{\text{N}}|},$$

[8] $|E_q^{\text{N}}|$ in the selection weight operates as a smoothing factor: Many of the words that do not match useful documents in E may do so in the collection at hand.

which we need to compute only over the useful documents. Similarly to [BYG11], we rely on an efficient estimator u_{U} of w_{U}, defined as:

$$u_{\mathrm{U}}(d,q) \triangleq \frac{\mathrm{NFE} \cdot \mathrm{IDE}(d) \cdot \mathrm{UE}(q) \cdot \pi_D(d)}{w \cdot |E_q^{\mathrm{U}}| + |E_q^{\mathrm{N}}|},$$

to keep estimation costs to reasonable levels. Here, NFE and UE(q) are estimators of normalization factor $\mathbb{Z}_w$ and number of useful documents retrieved by a query.

The key challenge in computing NFE is to account for the number of queries $|Q_+^{\mathrm{U}}|$ and the distribution of their selection weight. We can compute them both on the fly while sampling during our trial process: Both factors can be computed by sampling according to the selection weight, instead of uniformly at random, as for the PSE estimator in Bar-Yossef et al. [BYG11]. We compute NFE by keeping track of the fraction of sampled queries that retrieve at least one useful document, defined as α, and in turn computing $\mathrm{NFE} = \alpha \cdot \sum_{q \in Q}(w \cdot |E_q^{\mathrm{U}}| + |E_q^{\mathrm{N}}|)$.[9] To compute UE($q$), we proceed similarly to IDE computation Bar-Yossef et al. [BYG11]: We sample documents uniformly at random from C_q until we find a useful document; if we find the useful document after processing n documents, then $\mathrm{UE}(q) = \frac{|C_q|}{n}$.

By favoring queries likely to be useful our approach potentially address the limitations of random approaches. However, we still need to consider the selection of w, which controls the selection bias in our estimations. Choosing $w \approx 1$, will exhibit comparable limitations to the baseline random approach, while choosing $w \gg 1$ may lead to issuing very specific queries that may not be representative of the useful documents in the collection at hand. We experimentally show the impact of different values of w in Sections 5.5 and 5.6, and provide a guideline on how to select this value.

5.4.3 Targeted Query Pool-Free Estimator

As discussed in Section 5.3.3, the query-pool free approach in [ZZD13] may rarely find useful documents, because the random walk is performed over all queries, independently of the target function $f(d)$. To estimate the number of useful documents, we could restrict

[9] Both $|E_q^{\mathrm{U}}|$ and $|E_q^{\mathrm{N}}|$ values in this formula are computed once and for all during the generation of the queries.

the graph to only useful queries (i.e., queries that retrieve at least one useful document and for which $f(d) = 1$). Unfortunately, such graph could be largely disconnected and the random walk would be unable to fully explore it, thus leading to inaccurate usefulness estimations. Instead, we propose to keep the original graph—so that we can fully explore it—and modify the random walk process to favor visiting useful queries—so that we are more likely to observe documents for which $f(d) = 1$.

We define a *weighted graph*, on which we perform a "weighted" random walk (i.e., edge e with weight $w(e)$ is selected from an edge set E with probability $w(e)/\sum_{e' \in E} w(e')$). We define an edge (q, q') to be *useful* if and only if both queries it connects retrieve at least one useful document. We assign useful edges weight w and the others 1. In Equation (5.2) we thus replace the original (unweighted) degree $d(q)$ with the weighted counterpart: $d_w(q) = w \cdot N_{\text{U}} + N_{\text{N}}$, where N_{U} is the number of q's useful incident edges (i.e., useful neighbors q') and $N_{\text{N}} = N_{\text{ALL}} - N_{\text{U}}$ (with $N_{\text{ALL}} = d(q)$). The definition of $\omega(d)$ (i.e., the number of queries that retrieve it) remains unchanged, thus we still estimate it using the method in [ZZD13].

To estimate the weighted degree $d_w(q)$ of a sampled query q we need (i) the number of all incident edges N_{ALL}, and (ii) the number of useful edges N_{U}. For N_{ALL} we proceed as in [ZZD13] for the degree $d(q)$ (see Section 5.3.3). For N_{U} we proceed similarly, now counting the number of sampling attempts n_{U} we need to find a q' that both matches q and is useful, to estimate $\hat{N}_{\text{U}} = |Q'|/n_{\text{U}}$.[10] Thus, we calculate $\hat{d}_w(q) = w \cdot \hat{N}_{\text{U}} + (\hat{N}_{\text{ALL}} - \hat{N}_{\text{U}})$. The computation of $|V_c|$ uses the unweighted graph to collect the startup collection, as described for the baseline approach in Section 5.3.3.

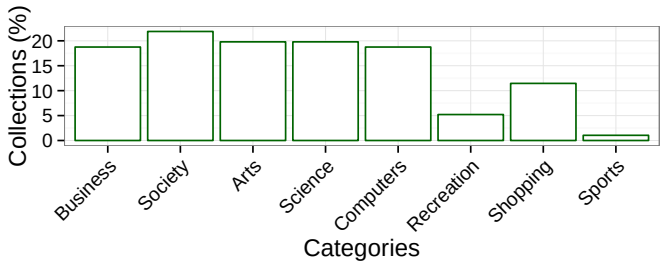
Thus far, we have described the (adaptations of) traditional approaches and information extraction-specific approaches that we study in this chapter. We now describe the settings for our experimental evaluation of these techniques (Section 5.5) and report our results (Section 5.6).

[10] Implementation-wise, we can get n_{ALL}, the sampling attempts for N_{ALL}, from the same sampling sequence as n_{U}.

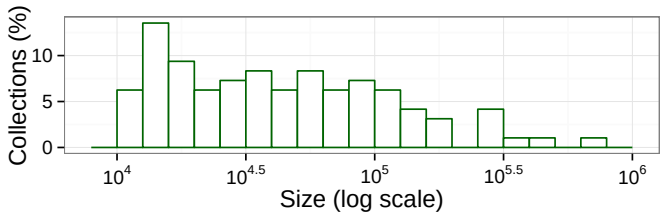
5.5 Experimental Settings

Collections: Our *test set* consists of 96 real web collections across different topics, collected using an approach similar to that in [GIS03] over the Open Directory Project (ODP) [ODP15]. Specifically, we first selected the 8 top-level ODP categories with the largest number of entries, namely, Business, Society, Arts, Science, Computers, Recreation, Shopping, and Sports. We then selected the 5 most popular subcategories in each of the 8 initial categories. In turn, we also picked the 5 most popular subsubcategories from each subcategory, for a total of 200 subsubcategories. For each subsubcategory, we then randomly chose 7 unique web collections that have a text search interface. Finally, we collected their contents using a state-of-the-art query-based sampler [CC01], issuing at most 20,000 queries and retrieving up to 1,000 documents for each. In our test set, we kept the collections that produced at least 10,000 documents following this method, to focus the evaluation on collections with a substantial number of documents. We show the distribution of covered categories and their sizes in Figure 5.3. Our *tuning set*, which we use for tuning parameters of the various techniques, consists of 40 collections selected randomly from among the collections under the above subsubcategories and not in the test set. We collected documents from these tuning collections using the Nutch Web crawler [Nut15]. We indexed each collection, in both the tuning set and the test set, with the text retrieval toolkit Lucene [Luc15], to emulate the query-only behavior of deep web collections and only access them through their query interface. (Fully-accessible collections can be indexed once and for all using Lucene, to provide query access.) We exhaustively processed the collections with our information extraction systems (see below) to obtain the real number of useful documents in each collection. We also used TREC 1-5 collections [TRE00] for different operations (e.g., query pool construction), which we describe as needed throughout this section.

Information Extraction Systems: We evaluated a variety of information extraction systems and components for all relations in our experiments (see below) via 5-fold crossvalidation over a set of training documents, and selected the two best-performing combinations, namely, Subsequence Kernel (SSK) [BM05b] and Bag of n-Grams Kernel (BONG) [GLR06]. We implemented them using REEL (see Chapter 3). We also considered different named



(a) Distribution of categories



(b) Distribution of collection sizes

Figure 5.3: Category distribution (a) and size distribution (b) of the test set collections.

entity taggers and selected: for *person* and *location* entities, the pretrained conditional random fields (CRF) [ML03] from the StanfordNLP package [sta15a]; for *natural disasters*, CRFs from the E-txt2DB framework [Etx12]; for the remaining entities, maximum entropy markov models (MEMM) [MFP00], also from E-txt2DB.

Relations: For a robust evaluation, we include 5 substantially different relations in the experiments, as noted in Table 5.2. Four such relations, namely, Natural Disaster–Location, Man Made Disaster–Location, Person–Charge, and Election–Winner, are sparse, in that very few documents tend to be useful for them. In contrast, relation Person–Career is a dense relation. (Table 5.2 shows the percentage of useful documents for each relation in the TREC 1-5 collections for the two information extraction systems. Also, Figure 5.4 shows the distribution of useful documents over the collections in our test set.)

Technique Tuning: We tuned each technique over the tuning set and using the SSK information extraction system over Man Made Disaster–Location.

Surrogate-based methods: The tuning of the surrogate-based methods was performed as follows:

- *Baseline (Section 5.3.1):* We evaluated different query sets Q and numbers of queries k. For Q, we generated one-word queries using the SVM approach in [MBGM04] and

Relation	Useful documents	
	SSK	BONG
Person–Career	56.20%	55.95%
Natural Disaster–Location	2.03%	2.74%
Man Made Disaster–Location (*)	0.80%	0.87%
Person–Charge	1.55%	1.84%
Election–Winner	0.24%	0.84%

Table 5.2: Fraction of useful documents found in the TREC 1-5 collections for relations extracted using two information extraction systems, SSK and BONG. We use (*) only during tuning.

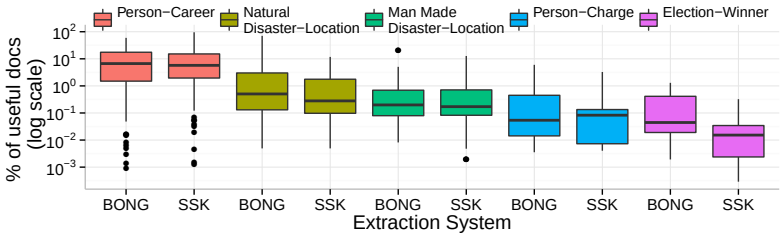


Figure 5.4: Fraction of useful documents for each relation across our 96 test collections. The box boundaries are the 25th and 75th percentiles, the bold horizontal line inside each box is the median, and the dots denote outliers.

two effective feature selection methods, namely, Information Gain and χ^2 test [ZWS04] (see learning-based query generation techniques in Section 4.3), over 10,000 documents (50% useful and 50% useless) from TREC 1-5; we kept the words that are discriminative of the useful class; also, and as suggested in [AG03], we evaluated a query set including all words in the documents and another one removing the tuple attribute values (see explicit and implicit candidate set of keywords in Section 4.3). We varied $k \in [10, 200]$ in intervals of 10. We built the document samples using the query-based sampling technique in [CC01], which produces nearly-random document samples from words collected from retrieved documents.

- *Information extraction-specific (Section 5.4.1):* We compared (i) the query-based sampling technique in ReDDE; (ii) the bootstrapping-based sampling approach in [AG03], which starts from a set of seed tuples as queries that is iteratively expanded as it collects more useful documents and extracts tuples from them (see Tuples in Section 4.3); (iii) Our Cyclic sampling technique (Section 4.2.3) that issues the queries in Q above

based on their score and processes the documents iteratively and in rounds. Finally, to find the useful terms and to asses term bias we used the Fisher exact test in [Fis36] varying p-value $\in [0.01, 0.1]$.

Query pool-based methods: The tuning of the query pool-based methods was performed as follows:

- *Baseline (Section 5.3.2):* We built two query pools of single words from TREC 1-5: (i) a generic query pool (G) of 4M words considering all documents; and (ii) an information extraction specific query pool (S) that only considers words from useful documents. We performed the estimation process with and without Rao-Blackwellization, suggested in the original paper for variance reduction [BYG11]. We evaluated the sum and average approaches.

- *Information extraction-specific (Section 5.4.2):* In addition to the configuration of Baseline above, we evaluated several different values for the weight $w \in [50, 500000]$. We also used the TREC 1-5 collections as our external collection and varied the number of documents to retrieve from it $k \in [10, 1000]$ for weight computation.

Query pool-free methods: The tuning of the query pool-free methods was performed as follows:

- *Baseline (Section 5.3.3):* We varied three parameters: (i) the number of sampled queries $|S| \in [10, 500]$, with increments of 10, (ii) the length of h-grams, $h \in \{1, 2, 3\}$, and (iii) the length of the burn-in of the random walk b before collecting the samples, $b \in [50, 500]$ with increments of 50.

- *Information extraction-specific (Section 5.4.3):* Besides the parameters for Baseline above, we evaluated several different values for its weight $w \in [10, 5000]$.

Techniques for Comparison: We compared the following alternatives over the 96 collections in our test set, with the settings derived via tuning, as summarized below:

- *Baseline surrogate-based (ReDDE):* We generate the samples with the query-based document sampler in [CC01], issuing up to 300 queries and collecting (at most) 5

documents for each. We use the queries learned with χ^2 for Q and use the top-100 queries (i.e., $k = 100$).

- *Baseline query-pool-based (PB):* We use the sum (ABS) and average (AVG) methods and perform Rao-Blackwellization. (The impact on efficiency of performing Rao-Blackwellization is low, because the number of documents processed with the information extraction system is relatively small.)

- *Baseline query-pool-free (PF):* We use an English dictionary to randomly find an initial query for the estimation process. We use single terms as queries (i.e., $h = 1$) and only accept queries that retrieve at least $u = 3$ documents, as suggested in [ZZD13]. We collect 100 queries for V_c, the startup query path that can be shared across information extraction tasks, and at most 100 for the estimation sample queries S.

- *Information extraction-specific surrogate-based method (Surrogate):* For sampling, we derive Q using the χ^2 method and excluding tuple attributes. We considered the useful queries in order of χ^2. Our sample S has at most 1000 documents. To select the useful terms T^{U}, we use p-value $= 0.05$ in the Fisher test.

- *Information extraction-specific query-pool-based method (PB-W):* As with the baseline, we evaluate sum (ABS) and average (AVG) estimators using the G and S query pools and applying Rao-Blackwellization. We index our external collection using Lucene with default parameters. We use $k = 1000$ for the query weight computation and $w = 5000$.

- *Information extraction-specific query-pool-free method (PF-W):* We use a similar configuration to that in PF, but with $w = 1000$.

Additional Settings: Estimators issue at most 100 queries, and retrieve up to 50 documents per query. To account for randomness, we run each estimator five times and report average values over the five runs. Finally, note that all estimators contain some form of (weighted) averaging of a metric over samples S (e.g., for the pool-free estimator, we calculate the individual contribution of each q in the summation in the denominator of Equation (5.2)). We filter outliers from this average, using the outlier detection algorithm

in [vdL10, "Method I"] as implemented in the R [RSo15] package "extremevalues:" a value x is an outlier if it is outside the limit where less than 1 observation is expected, based on observed data within quantile limits $[\alpha, 1-\alpha]$. We evaluated every estimator with and without outlier removal (using $\alpha \in \{0.05, 0.1\}$). Removing outliers using $\alpha = 0.1$ performed best across all techniques. Thus, our final results include such removal.

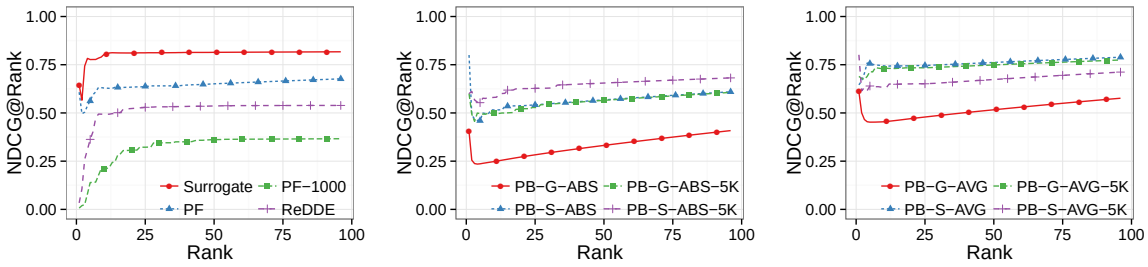
Evaluation Metrics: We measure *ranking quality* and *estimation cost* with the following metrics:

- *Cumulative Gain (CG@k):* We measure the number of useful documents that we obtain by processing the collections in ranking order. If u_i is the number of useful documents in the ith collection, we define $\text{CG@}k = \sum_{i=1}^{k} u_i$. This metric focuses on absolute values of useful documents, which makes the comparison across relations problematic, and also does not fully capture "errors" in the ranking.

- *Normalized Discounted Cumulative Gain (nDCG@k):* nDCG@k alleviates the limitations above in a robust manner. Specifically, nDCG@k is defined as the normalized version of Discounted Cumulative Gain $\text{DCG@}k = u_1 + \sum_{i=2}^{k} \frac{u_i}{log_2(i)}$, which penalizes the errors in the ranking order.[11] Now, to normalize DCG@k (and obtain nDCG@k), we need to calculate the DCG@k of an ideal ranking, namely, IDCG@k. Finally, $\text{nDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}$.

- *Processed Documents (PD) and Issued Queries (IQ):* We measure the efficiency of the information extraction process in terms of the number of issued queries and processed documents, and not running time. The reasons for this are twofold: (1) many factors (e.g., network traffic, collection responsiveness) can distort running times in the distributed environments on which we focus and are difficult to capture reliably; and (2) the number of issued queries and processed documents are good indicators of expected running time. Finally, we report these values only for the actual estimation process and ignore the initial, once-and-for-all processing (e.g., collection size estimation or random document sample generation) required by the techniques, which gets

[11] Variants of DCG@k exist in the literature; the version we use accounts for the distribution of useful documents.



(a) Surrogate and pool-free (b) Pool-based (sum) (c) Pool-based (avg)

Figure 5.5: nDCG@k for Natural Disaster–Location, for the BONG information extraction system and issuing (at most) 100 queries.

amortized over time.

- *Relative Estimation Error (RE):* We measure the relative estimation error, defined as $\text{RE}(C) = \frac{||\hat{C^{\text{U}}}|-|C^{\text{U}}||}{|C^{\text{U}}|}$, to assess the accuracy of the estimators. Because we compute our estimations issuing a number of queries (al least) an order of magnitude smaller than those reported in earlier work, we expect the estimation error to be higher than those reported in the literature.

- *Non-Zero Estimates (NZ):* We measure the fraction of estimates that produce non-zero estimates, to assess the impact of finding useful documents during the estimation. With this metric we can thus better explain the performance of different estimators (e.g., if they find many useful documents but fail to produce accurate estimates).

5.6 Experimental Results

We now evaluate the baseline and information extraction-specific ranking approaches of Sections 5.3 and 5.4, with the settings of Section 5.5.

5.6.1 Quality of Collection Ranking Approaches

We evaluate the ranking approaches over all relations and information extraction systems of Section 5.5. Figure 5.5 shows nDCG@k of all techniques over the entire rank of collections (i.e., $k \in [1, 96]$) for Natural Disaster–Location using the BONG information extraction

system, and issuing at most 100 queries. (Other relations and extraction systems—with the exception of Person–Career, which we study in detail later—yielded similar results. Also, we later vary the number of issued queries.) Figure 5.5a, for the surrogate and query pool-based methods, shows that the PF baseline outperforms PF-1000, its information extraction-specific counterpart, by almost 75%. PF-1000 requires on average more queries than PF to walk the random graph; for this reason, PF-1000 will rarely find useful documents at such small query budget. As we will see, when PF-1000 finds useful documents, its performance improves considerably, always overcoming its baseline counterpart. In contrast, among the surrogate methods, Surrogate outperforms ReDDE by almost 50%: the information extraction-specific document sample, although small, manages to include useful documents; also, ReDDE's collection descriptors do not accurately characterize the useful documents.

Figures 5.5b and 5.5c, for the query pool-based methods for sums and averages, show that the information extraction-specific versions also outperform the baseline counterparts. For sum, this difference is mainly based on the number of useful documents sampled during the estimation, because there are more non-zero components to include in the estimation. For this reason, PB-S-ABS-5K is best, with its weighted specific query pool highlighting potentially useful queries. For average, the quality of the ranking also depends on finding queries that retrieve a combination of useful and useless documents, as both types are crucial for computing the average in question. PB-S-AVG and PB-G-AVG-5K sample such queries and thus exhibit the highest quality in this family. Overall, and across families, the top contenders are Surrogate and the average pool-based estimators, which effectively exploit both useful and useless documents during estimation.

In addition to the (ranking-based) evaluation above, we also evaluate the quality of our estimators by computing their relative estimation error. Table 5.3 shows the relative estimation error of our estimators for the Natural Disaster–Location relation and using the BONG information extraction system. (Other relations and information extraction systems yielded similar conclusions. Also, this table does not include ReDDE, because it is not an estimator of the number of useful documents in a collection.) Rather than computing an aggregated measure over all collections, we report the relative estimation error for different collection splits, according to their number of useful documents: We produce the collection

Technique name	Splits [Min - Max]				
	[1-1]	[6-80]	[97-1409]	[1708-32301]	[93235-144457]
Surrogate	**1.00**	**0.66**	**0.63**	**0.74**	0.64
PF	1.15	1.15	1.43	0.99	0.97
PF-1000	1.96	5.11	4.97	1.03	0.98
PB-G-ABS	**1.00**	4.23	1.07	1.00	0.99
PB-G-AVG	**1.00**	62.22	3.28	1.25	0.89
PB-S-ABS	1.50	1.59	0.97	0.97	1.00
PB-S-AVG	17.91	12.85	5.74	1.19	**0.53**
PB-G-ABS-5K	1.00	1.03	0.93	0.98	1.00
PB-G-AVG-5K	**1.00**	5.12	9.59	1.33	0.54
PB-S-ABS-5K	3.17	1.41	0.92	0.97	0.99
PB-S-AVG-5K	37.20	15.22	5.48	1.05	0.60

Table 5.3: Relative estimation error for Natural Disaster–Location, using the BONG information extraction system and issuing (at most) 100 queries.

splits using log-scaled usefulness values (i.e., we use the logarithm of the number of useful documents in a collection $\log_{10}|C^{\mathrm{U}}|$ instead of $|C^{\mathrm{U}}|$ directly) to effectively cover the large range of possible usefulness values. For clarity, Table 5.3 shows the minimum and maximum number of useful documents in each split. As shown, relative errors tend to be high: this is due to the low number of queries that we issue to meet our efficiency constraints. By issuing thousands of queries—which is highly impractical for our information extraction setting—we can reduce the relative estimation error significantly. We also observe that most techniques tend to perform better for collections with a large number of useful documents. Finally, based on this analysis, we conclude that Surrogate consistently produces the most accurate estimations.

To understand the actual number of useful documents observed as we process collections in ranking order, Figure 5.6 shows CG@k for Natural Disaster–Location and Person Career using the BONG information extraction system, for a selection of high-quality techniques according to our experiments. For reference we also show the CG@k of an ideal ranking (labeled Ideal) and of a random ranking (noted as a dashed line). Figure 5.6 reveals substantial gains from prioritizing the collections for extraction. For Natural Disaster–Location, for example, Surrogate effectively identifies the collections containing 95% of the total useful documents within the top-10 collections. This would translate to an efficiency improvement

of almost 90% if we were to only process these top-10 collections, because we would ignore 86 (out of 96) test collections. A noticeable benefit is also shown for Person–Career, which we discuss in detail later in this section, for Surrogate, ReDDE, and PB-G-AVG-5K.

5.6.2 Efficiency of Collection Ranking Approaches

To understand the efficiency of the different approaches, we analyze the extraction cost and achieved ranking quality at different stages in the estimation process. Figures 5.7 and 5.8 show, respectively, the number of documents processed with the information extraction system and nDCG@10 for different numbers of issued queries, for Person–Charge using the SSK information extraction system. (Different relations and information extraction systems yielded analogous conclusions.) We show the same technique splits as in the above analysis, for clarity. Figure 5.7 shows that information extraction-specific techniques process on average more documents than their baseline counterpart. ReDDE is an exception: it does not incur querying or extraction costs during the estimation process.

The reasons as to why information extraction-specific approaches process more documents on average are manifold and differ for each family of techniques. Notably, for Surrogate (Figure 5.7a), the number of extracted documents grows with the size of the document sample, because all sampled documents need to be processed. For the pool-free family (also in Figure 5.7a), PF-1000 may need to process several documents before choosing the next "hop," due to its weighted walking strategy. In contrast, PF does not need this operation, since it navigates the graph only based on the document contents, without incurring additional extractions. However, these techniques may retrieve several hundred documents from each collection for graph construction, to extract h-grams from the documents.

Finally, for the pool-based families (Figures 5.7b and 5.7c), the extraction cost grows with the number of observed useful documents (from the sampled edges). In fact, after obtaining a useful document further operations take place: (i) for sum, the inverse degree estimator issues additional queries to the collection, although it avoids processing the documents with the information extraction system; the average estimators do not need inverse degree estimation and, therefore, more documents are often processed for the same query budget; and (ii) the Rao-Blackwellization method for variance reduction processes all doc-

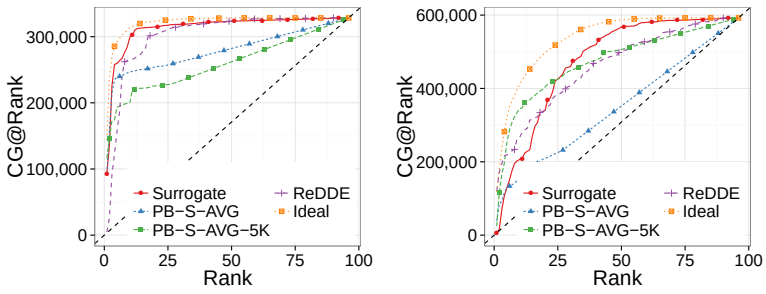
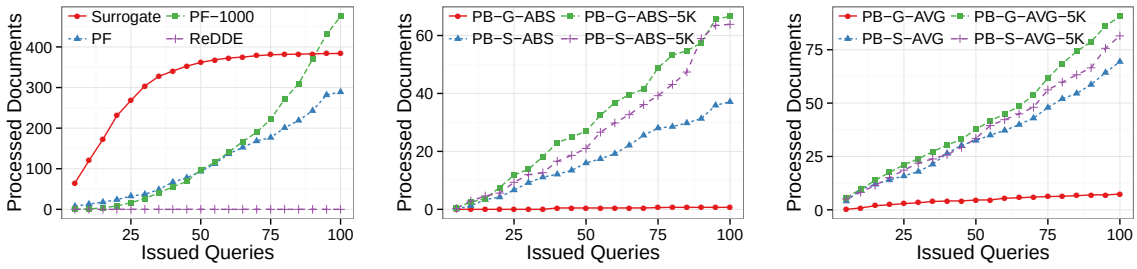
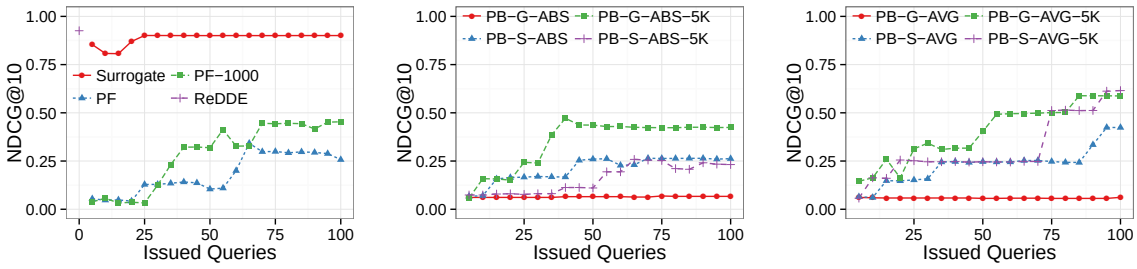


Figure 5.6: CG@k for Natural Disaster–Location (left) and Person–Career (right) for the BONG information extraction system and issuing (at most) 100 queries.



(a) Surrogate and pool-free (b) Pool-based (sum) (c) Pool-based (avg)

Figure 5.7: Processed documents for Person–Charge, for the SSK information extraction system and different numbers of issued queries.



(a) Surrogate and pool-free (b) Pool-based (sum) (c) Pool-based (avg)

Figure 5.8: nDCG@10 for Person–Charge, for the SSK information extraction system and different numbers of issued queries.

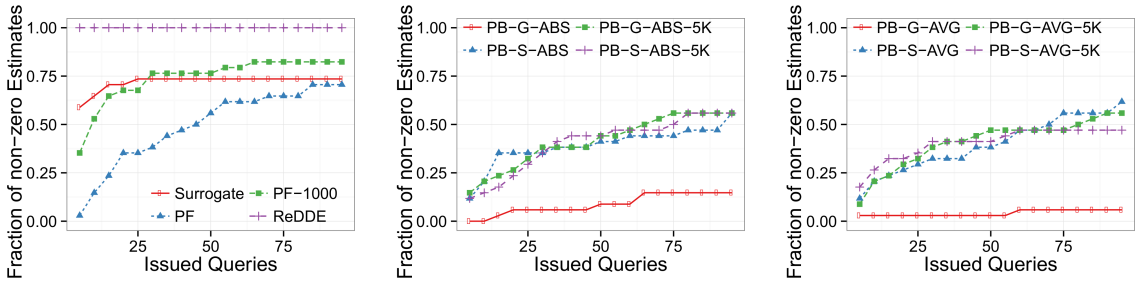
uments that the current query retrieves; this increases the extraction cost, but this cost is comparable across the techniques we evaluated.

After analyzing the extraction cost, we also study the ranking quality associated with such costs. Figure 5.8 shows that the quality of the ranking improves with the number

of issued queries: The estimators learn more—and more reliably—about the collections as the estimation progresses, which is reflected in better estimations. These improvements, though, vary considerably across techniques: Surrogate, for instance, produces high-quality estimates even when issuing a small number of queries, whereas the remaining techniques improve their quality progressively, with a steeper gain for information extraction-specific than for baseline approaches. Furthermore, some information extraction-specific techniques achieve higher quality rankings than other techniques at a fraction of their cost. For example, after issuing 25 queries, PB-G-ABS-5K and PB-G-AVG-5K exhibit comparable quality to PB-S-ABS and PB-S-AVG after issuing 100 queries, respectively.

5.6.3 Support of Collection Ranking Approaches

As discussed in Section 5.2, one of the crucial requirements of accurate and low-variance estimation methods is producing non-zero terms (i.e., terms in which $f(d) \neq 0$), to in turn produce non-zero estimates. In our information extraction setting, this implies obtaining useful documents—for which $f(d) = 1$—along the estimation process. We now analyze the fraction of non-zero estimates that different estimation methods produce, computed over the five independent runs that we perform for each method (see Section 5.5). Figure 5.9 shows the fraction of non-zero estimates of all techniques for Election–Winner using the BONG information extraction system, and for varying numbers of issued queries. (Other relations and extraction systems yielded similar results.) As shown, baseline approaches, namely, PF (Figure 5.9a), PB-G-ABS (Figure 5.9b), and PB-G-AVG (Figure 5.9c), produce a significantly smaller fraction of non-zero estimates than information extraction-specific approaches, for the same number of issued queries. This corroborates our hypothesis of Section 5.2 and explains the poor quality that these methods exhibit for our ranking problem. Another important observation is that pool-based approaches are many times (about 25% of the time) unable to produce non-zero estimates: This occurs because the sampled queries rarely match—and retrieve—any documents from the collections. We observe this for PB-S-ABS-5K and PB-S-AVG-5K, two methods that heavily favor picking queries—from an information extraction-specific query pool—that are likely to retrieve useful documents. Next, we analyze the impact of different selection weights in the quality of the estimation



(a) Surrogate and pool-free (b) Pool-based (sum) (c) Pool-based (avg)

Figure 5.9: Fraction of non-zero estimates for Election–Winner, for the BONG information extraction system and different numbers of issued queries.

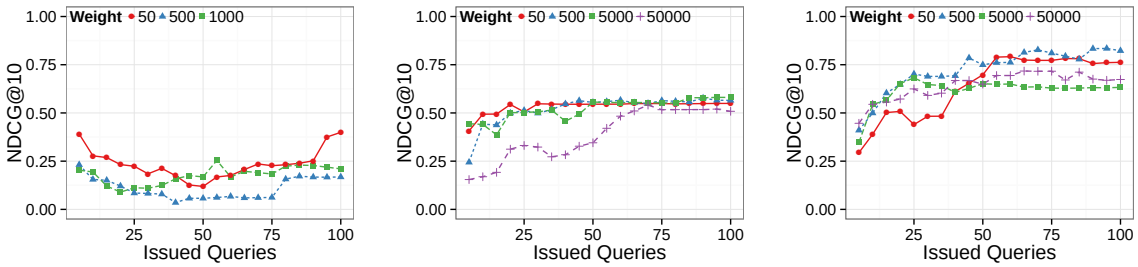
methods.

5.6.4 Impact of Selection Weight

We finally analyzed the impact in collection ranking of the selection weight that we use in our information extraction-specific estimators. Figure 5.10 shows nDCG@10 for PF-W (Figure 5.10a), PB-S-ABS (Figure 5.10b), and PB-S-AVG (Figure 5.10c), for Natural Disaster–Location using the BONG information extraction system and for varying numbers of issued queries. (Conclusions were analogous across relations and within techniques of the same families.) As shown, increasing the selection weight has limited impact on the quality of the ranking. In fact, using high selection weights (e.g., see Weight=50000 in Figures 5.10b and 5.10c) hurts the quality of the ranking because the selected queries are too specific to the training collection and do not generalize well to other collections. For the pool-free approach (see Figure 5.10a), the low performance of high selection weights is due to the cost of "walking" the weighted random graph: In this approach, higher selection weights indicate a higher number of issued queries and processed documents.

5.6.5 Impact of Collection Characteristics

Text collections are rather heterogeneous, with substantial differences in size and contents. Notably, large collections are problematic for baseline approaches, as Figure 5.6a shows for Natural Disaster–Location, where the most useful collections were among the largest collections (350,000 documents on average). We showed the effectiveness of our information



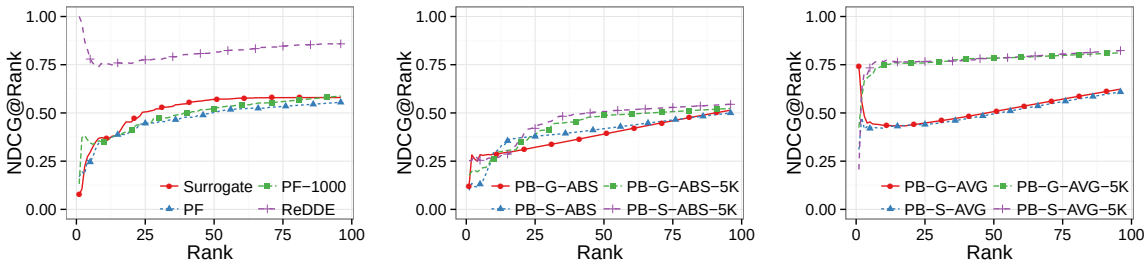
(a) Pool-free (b) Pool-based (sum) specific pool (c) Pool-based (avg) specific pool

Figure 5.10: nDCG@10 for Natural Disaster–Location, for the BONG information extraction system and different numbers of issued queries.

extraction-specific approaches for this case. Similarly, the collection contents also affect some of the other approaches: Pool-based approaches retrieved substantially fewer documents than pool-free approaches (see Figure 5.7), because many queries in the query pool were not topically relevant to the contents of the collections; pool-free approaches do not exhibit this problem.

5.6.6 Impact of Information Extraction-task Characteristics

To understand the impact of relation characteristics, we now focus on a dense relation. Figure 5.11 shows nDCG@k over the entire rank of collections (i.e., $k \in [1, 96]$) for Person–Career using the BONG information extraction system, and for all techniques. We show the same technique splits as in the above analysis, for clarity. Figure 5.11a shows that Surrogate, PF, and PF-1000 exhibit comparable (low) quality: these techniques failed to correctly rank large collections with a large number—but a relatively low fraction—of useful documents. Specifically, Surrogate was unable to obtain discriminative terms for the estimation, since most sampled documents were useful. The pool-free approaches, on the other hand, were unable to reach many useful documents, because the useful documents in the top collections were a small fraction in each collection. The pool-based sum estimators in Figure 5.11b exhibit a trend similar to that of pool-free approaches. However, two sets of techniques performed relatively well, namely, ReDDE (Figure 5.11a) and the information extraction-specific pool-based AVG estimators (Figure 5.11c). These techniques benefited from the scaling to the entire collection, because the largest collections were among the top most



(a) Surrogate and pool-free (b) Pool-based (sum) (c) Pool-based (avg)

Figure 5.11: nDCG@k for Person–Career, for the BONG information extraction system and issuing (at most) 100 queries.

useful collections.

5.6.7 Additional Discussion

An orthogonal, interesting aspect to the discussion above concerns the overhead incurred by a ranking approach when new collections or information extraction tasks arrive. Table 5.1 summarized the collection- and information extraction-specific requirements of each ranking technique. For new collections, almost all techniques require some preprocessing (see collection column in Table 5.1). Deciding the approach to adopt will not only depend on the performance and overhead of the techniques but also on the number of information extraction systems that we will run over each (new) collection. Because size estimation and graph construction may take up to several thousand queries, approaches that rely on these will only be reasonable when many information extraction systems are involved, as the cost will amortize over time. In other cases, though, estimators that do not require additional information from the collection, such as pool-based estimators for sums, may be the best choice.

Similarly, for new information extraction tasks, almost all techniques also require some preprocessing (see information extraction column in Table 5.1). The most expensive process is, by far, producing a specific query pool (with or without query weights). The remaining processes, namely, learning queries for document sampling or for querying the descriptor in ReDDE and producing a query sample for the pool-free techniques, are relatively inexpensive. Therefore, and given our quality and efficiency results, surrogate methods (i.e.,

ReDDE or Surrogate) seem to be the most reasonable choice for new information extraction tasks. However, if new collections are expected to appear at high rates, it may be worthwhile building—and amortizing the construction of—a query pool, so that the estimation starts without overhead.

5.7 Conclusions

In this chapter, we introduced and addressed the problem of ranking text collections for an information extraction task, to prioritize the extraction effort by focusing on collections with substantial numbers of useful documents for the information extraction task. Specifically, the problem is that of effectively and efficiently ranking a set of text collections according to their number of useful documents for a given information extraction task. We studied both (adaptations of) state-of-the-art resource selection strategies, and information extraction-specific approaches. We performed an extensive experimental evaluation over realistic Web collections, and for several different information extraction tasks. Our evaluation focused on the quality and efficiency characteristics of the ranking approaches, with the following conclusions: (1) we found top contenders for each of these characteristics and provided insight on how to choose among them; (2) we analyzed which techniques are better suited for certain characteristics of the collections, such as their size and contents, and of the information extraction tasks, such as their sparsity; and (3) we discussed the overhead incurred by each technique in dynamic domains, where new collections or information extraction tasks may arrive.

The main contributions of this chapter are the introduction and extensive evaluation of the problem of ranking text collections for information extraction. We believe that the approaches in this chapter can serve as the basis for future efforts on prioritizing the extraction process over large volumes of text. As we will see in our future work discussion (Chapter 10), for instance, our approaches provide the necessary infrastructure to support—with rather minimal intervention—collection ranking along other relevant dimensions (e.g., quality of the extracted tuples). Overall, this chapter provides a roadmap for addressing this critically important building block for efficient, scalable information extraction.

Chapter 6

Ranking Documents for Scalable Information Extraction

In Chapter 5, we discussed how we can prioritize useful collections for an information extraction task of interest, so that we can focus the extraction effort, for efficiency. Moreover, we showed that the useful documents—even in these useful collections—are only a (very) small fraction of the entire collections. If we could effectively identify the set of useful documents in a collection, we would complete the extraction process while decreasing the extraction time considerably without any need to change the information extraction system.

In this chapter, we present an adaptive document ranking approach to effectively and efficiently prioritize the useful documents in a collection for an information extraction task of interest. Specifically, we propose a principled, efficient learning-to-rank approach that prioritizes documents for an information extraction task by combining: (i) online learning [SSSS07], to train and adapt the ranking models incrementally, hence avoiding computationally expensive retrains of the models from scratch; and (ii) in-training feature selection [GE03], to identify a compact, discriminative set of words and phrases from the documents to train ranking models effectively and efficiently. Importantly, our approach revises the document ranking decisions periodically, as the ongoing extraction process reveals (fine-grained) characteristics of the useful documents for the extraction task at hand. Our approach thus manages to capture, progressively and in an adaptive manner, the het-

erogeneity of language and content typically exhibited by the useful documents, which in turn leads to information extraction executions that are substantially more efficient—and effective—than those with state-of-the-art approaches, as we will see.

In summary, the main contributions of this chapter are:

- An end-to-end document ranking approach for effective and efficient information extraction in an adaptive, online, and principled manner (Section 6.2). We include two low-overhead ranking algorithms for information extraction based on learning-to-rank strategies. These algorithms perform online learning and in-training feature selection (Section 6.2.1). In addition, we present two techniques to detect when adapting the ranking model for information extraction is likely to have a significantly positive impact on the ranking quality (Section 6.2.2).

- An experimental evaluation of our approach using multiple extraction tasks implemented with a variety of extraction approaches (Sections 6.3 and 6.4). Our approach has low overhead and manages to achieve higher accuracy than the state-of-the-art approaches, and hence is a substantial step towards scalable information extraction.

We now review necessary background and define our problem of focus in this chapter (Section 6.1). The bulk of this chapter has been published as [BSGG15].

6.1 Background and Problem Definition

Because information extraction systems are computationally expensive, as argued in Chapter 2, processing all documents exhaustively becomes prohibitively time consuming for large document collections. Ideally, we should focus the extraction effort on the useful documents for the information extraction system at hand.

As a crucial task, information extraction optimization approaches (e.g., Holistic-MAP [SGG13]) choose a document selection strategy to identify documents that are likely to be useful. State-of-the-art approaches for such document selection (e.g., QXtract [AG03], PRDualRank [FC11], and FactCrawl [BLNP11a]) are based on the observation that useful

documents for a specific relation[1] tend to share distinctive words and phrases. Discovering these words and phrases is challenging because: (i) many extraction systems rely on off-the-shelf, black-box components (e.g., named entity recognizers), from which we cannot extract relevant words and phrases directly; and (ii) machine learning techniques for information extraction do not generally produce easily interpretable models, which complicates the identification of relevant words and phrases. QXtract learns these words and phrases through document classification: after retrieving a small document sample, QXtract automatically labels each document as useful or not by running the extraction system of interest over these documents. QXtract can thus learn that words like "richter" or "hypocenter" are characteristic of some of the useful documents for the *Occurs-in* relation. Then, QXtract uses these learned words and phrases as keyword queries to retrieve (other) potentially useful documents (see Figure 6.1). More recent approaches (e.g., FactCrawl [BLNP11a] and PRDualRank [FC11]) adopt similar retrieval-based document selection strategies.

QXtract issues queries to the standard keyword search interface of document collections in order to retrieve potentially useful documents for extraction. Such keyword search interface, unfortunately, is not tailored for information extraction: the documents that are returned for a keyword query are ranked according to how well they match the query and not on how useful they are for the underlying information extraction task [BLNP11a]. For example, the query [tornado] for the *Occurs-in* relation returns only 145 useful documents among the top-300 matches from our validation split of the New York Times annotated corpus [NYT15] (see Section 6.3) using Lucene [Luc15], a state-of-the-art search engine library.

FactCrawl [BLNP11a] moves a step beyond keyword search and re-ranks the retrieved documents to prioritize the extraction effort (see Figure 6.1). Specifically, FactCrawl scores documents proportionally to the number and quality of the queries that retrieve them. FactCrawl determines the quality of each learned query—and of the query generation method that was used to generate the query—in an initial step, once and for all, by retrieving a small number of documents with the query and running them through the extraction

[1] Our approach is not applicable over open information extraction scenarios (e.g., [BCS+07]) where documents frequently contribute tuples to the open-ended extraction task.

system in question. With this initial step, FactCrawl derives: (i) for each query q, the F-measure $F_\beta(q)$, where β is a parameter that weights precision over recall; and (ii) for each query generation method m, the average $F_\beta^{avg}(m)$ of the F_β value of all queries generated with method m. During the extraction process, after retrieving documents with a set Q_d of queries learned via a query generation method m, FactCrawl re-ranks the documents according to a scoring function $S(d) = \sum_{q \in Q_d} F_\beta(q) \cdot F_\beta^{avg}(m)$. FactCrawl's document re-ranking process improves the efficiency of the extraction, since the documents more likely to be useful are processed earlier. However, FactCrawl exhibits two key weaknesses: (i) for document retrieval and ranking, just as QXtract (see discussion above), FactCrawl relies on queries derived, once and for all, from a small initial document sample, and hence may miss words and phrases relevant to the information extraction task at hand; and (ii) for document ranking, FactCrawl relies on a coarse, query-based document scoring approach that is not adaptive, and hence does not benefit from the wealth of information that is captured as the extraction process progresses.

Based on the discussion above, we now present our problem of focus in this chapter:

Problem Definition 3 *Consider a set of text documents D and an information extraction system E trained to extract tuples for a relation from text. Our goal is to prioritize the extraction effort of E over the documents in D, so that we process the useful documents for E earlier in the extraction process, for efficiency. Importantly, we want our ranking of documents to exhibit high precision and recall—to minimize the number of useless documents to process with E—while satisfying certain efficiency requirements (e.g., that running E over D in ranked order leads to a larger number of tuples faster than running E over D directly). Moreover, we want the document ranking to adapt to the relevant information about the extraction task (e.g., the real usefulness of the documents) obtained as E runs over documents from D.*

Adaptive models have been used for information extraction in a variety of ways. Early influential systems for large-scale information extraction, such as DIPRE [Bri98] and Snowball [AG00], have relied on bootstrapping to adapt to newly discovered information. Starting with a small number of "seed" tuples for the extraction task of interest, these systems

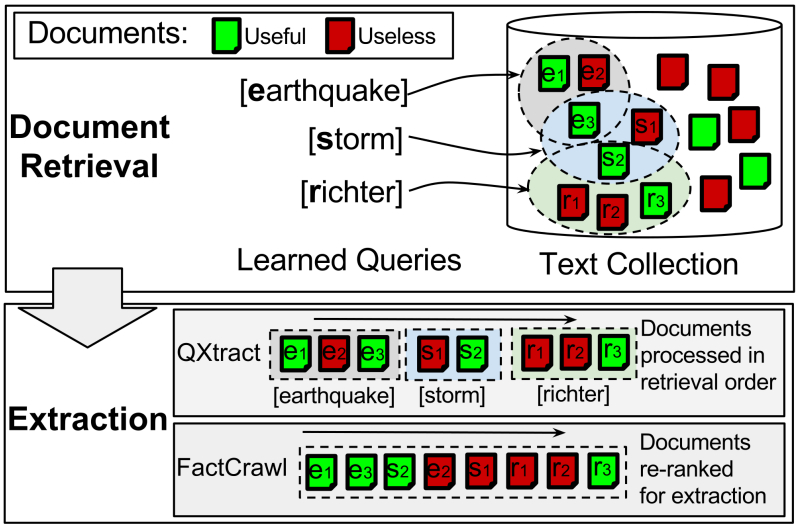


Figure 6.1: QXtract and FactCrawl.

learn and iteratively improve extraction patterns and, simultaneously, build queries from the tuples that they discover using these patterns. However, these systems are not suitable for our problem for two main reasons. First, techniques based on bootstrapping often exhibit far-from-perfect recall, since it is difficult to reach all tuples in a collection by using previously extracted tuples as queries [AG03; IAJG07]. Second, extraction systems are many times "black box" systems, which impedes the alteration of their extraction decisions. Other approaches (e.g., [CJTN06]) have relied on label propagation: starting with labeled and unlabeled examples, these approaches propagate the given labels to the unlabeled examples based on some example similarity computation. Such label propagation approaches are not beneficial for our extraction scenario, where the extraction system has already been trained and we can obtain new labels (i.e., useful or not) for previously unseen documents automatically by running the extraction system over them.

6.2 Online Adaptive Ranking

We now propose an end-to-end document ranking approach for scalable information extraction (see Figure 6.2) that addresses the limitations of the state of the art. Our approach prioritizes documents for an information extraction task—with a corresponding already-

trained information extraction system—based on principled, efficient learning-to-rank approaches that exploit the full contents of the documents (Section 6.2.1). Additionally, our approach revises the ranking decisions periodically as the extraction process progresses and reveals (fine-grained) characteristics of the useful documents for the extraction task at hand (Section 6.2.2). Our approach thus manages to capture, progressively and in an adaptive manner, the heterogeneity of language and content typically exhibited by the useful documents, which leads to extraction processes substantially more efficient—and effective—than those with state-of-the-art approaches, as we will show experimentally in Sections 6.3 and 6.4.

6.2.1 Ranking Generation

To prioritize the information extraction effort, by focusing on the potentially useful documents for the extraction system at hand, we follow a learning-to-rank approach (see Ranking Generation step in Figure 6.2). Similarly to state-of-the-art query-generation and ranking efforts (see Chapter 2), we obtain a small document sample and automatically "label" it with the information extraction system, without human intervention. We use the documents in this sample, with their words as well as the attribute values of tuples extracted from them as features, to train an initial document ranking model. After the initial document ranking is produced, we start processing documents, in order, with the information extraction system (see Tuple Extraction step in Figure 6.2).[2] Unfortunately, the initial ranking model is generally far from perfect, because it is learned from a necessarily small document sample. So our approach periodically updates and refines the ranking model (see Update Detection step in Figure 6.2), as new documents are processed and the characteristics of the useful documents are revealed, as we will discuss in detail in Section 6.2.2.

Unfortunately, state-of-the-art approaches for learning to rank [Liu09] are problematic for our document ranking setting for two main reasons. First, such approaches tend to be computationally expensive [Scu09], so updating and revising the ranking model contin-

[2]The pool of documents to process is either the full document collection, for collections of moderate size over which we have full access, or, alternatively, the documents retrieved with queries learned from the document sample. In Sections 6.3 and 6.4, we discuss this issue for our experiments further and experimentally study these two scenarios, which we introduced in Section 2.2.

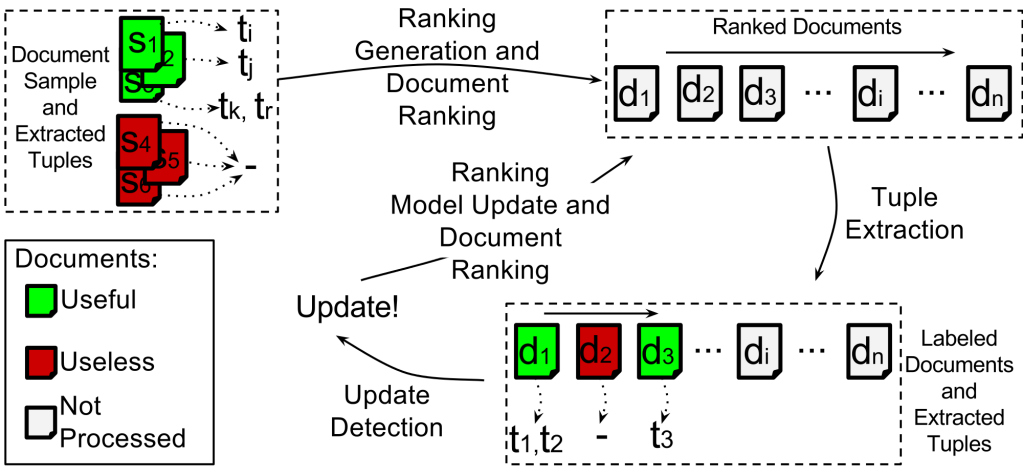


Figure 6.2: Our adaptive learning-to-rank approach for information extraction.

uously over time, as new documents are processed, would result in an unacceptably high overhead in the extraction process. Second, such approaches tend to require a relatively small feature space [BWG+10]. In contrast, in our ranking setting the feature space, including the document words and attributes of extracted tuples, is vast; furthermore, the feature space continues to grow as new documents are processed. Therefore, we need to develop unconventional learning-to-rank techniques for our ranking problem, to address the above two limitations of state-of-the-art approaches in an effective and efficient manner and without compromising the quality of the ranking models that we produce.

To address the efficiency limitation of learning-to-rank approaches, and to update the document ranking model efficiently, we rely on *online learning* [Bot10]. Using online learning, we can train the ranking model incrementally, one document at a time. Therefore, we can continuously adapt the ranking model as we process new documents, without having to retrain it from scratch. To adapt online learning to our problem, the main challenge is to define an update rule for the model—to be triggered when we observe new documents along the extraction process—that is simple enough to be efficient but, at the same time, sophisticated enough to produce high-quality models. From among the most robust online learning approaches [Bot10], the updates based on Pegasos gradient steps [SSSS07] are particularly well suited for our approach because of their efficiency and accuracy. Specifically, Pegasos gradient steps provide update rules that guarantee that learning techniques based on Sup-

port Vector Machines (SVM), the basis for some of the best-performing learning-to-rank approaches, learn high-quality models efficiently.

To address the feature-set limitation of learning-to-rank approaches, and to handle large (and expanding) feature sets, we rely on *in-training feature selection* [GE03]. In a nutshell, with in-training feature selection the learning-to-rank algorithm can efficiently identify the most discriminative features, out of a large and possibly expanding feature set, during the training of the document ranking model and without an explicit feature selection step. To do so, we rely on a sparse representation of the vectors that represent the feature weights, to discard all features with zero value. Therefore, our objective is to penalize models that rely on a large number of features with non-zero weight. Interestingly, we can rely on *regularization* [Bis06] to control the feature weight distribution in our learned models: regularization penalizes models that have undesirable properties such as having many features with non-zero weights, so we can use it for in-training feature selection and also to avoid overfitting. In our approach, we rely on a linear combination of two regularization methods, usually called elastic-net regularization [ZH05], which integrates: (i) the ℓ_1-norm regularization [TTA09], which tends to learn models where only a small subset of the features have non-zero weights; and (ii) the ℓ_2-norm regularization, which produces high-quality models by avoiding overfitting. This combination is necessary because the ℓ_1-norm regularization does not perform well when the number of documents is smaller than the feature space [ZH05], which is the case during early phases of the extraction process.

We now propose two learning-to-rank strategies, BAgg-IE and RSVM-IE, that overcome the limitations of state-of-the-art learning-to-rank approaches by integrating online learning and in-training feature selection, as discussed above.

6.2.1.1 BAgg-IE: A Pointwise Ranking Approach

Our first strategy, namely, *BAgg-IE*, incorporates online learning and in-training feature selection into a binary classification scheme where documents are ranked according to their assigned label and prediction confidence. Since binary classifiers optimize the accuracy of label assignment instead of the instance order, they are not optimized for ranking tasks [HGO00].

For this reason, BAgg-IE adopts a more robust approach that exploits multiple binary classifiers based on bootstrapping aggregation, or bagging [Bre96]. With this approach, the label assignments and confidence predictions derive from the aggregation of the answers of a committee of classifiers, rather than from an individual classifier. The intuition behind BAgg-IE is that each classifier is able to evaluate distinct aspects of the documents, thus collectively mitigating the limitations of each individual classifier. We adapt SVM-based binary classifiers [Joa98b] to support online learning and in-training feature selection. For online learning, our algorithm is based on Pegasos, in which each text document is a training instance and, hence, we update the model one document at a time. For in-training feature selection, each classifier in BAgg-IE combines the SVM binary classification problem with the regularization components of the elastic-net regularization framework that we discussed earlier, thus yielding the following learning problem to solve:

$$\arg\min_{\mathbf{w},b} \lambda_{All} \cdot \left(\frac{\lambda_{L2}}{2} \|\mathbf{w}\|_2 + (1 - \lambda_{L2}) \|\mathbf{w}\|_1 \right) + \sum_{(\mathbf{d},y)\in\mathcal{S}} \ell(y\langle \mathbf{w}, \mathbf{d} \rangle + b)$$

where b is the bias term, $y\langle \mathbf{w}, \mathbf{d} \rangle$ is the dot product of $\mathbf{w}$ and $\mathbf{d}$, ℓ is the hinge loss function, $\ell(t) = max(0, 1-t)$, and $\|\mathbf{w}\|_1$ and $\|\mathbf{w}\|_2$ are the ℓ_1 and ℓ_2-norms of the weight vector (i.e., the regularization components), respectively. Moreover, λ_{All} is the parameter that weights the regularization component over the loss function, and λ_{L2}, $0 \leq \lambda_{L2} \leq 1$, is the parameter that weights the ℓ_2-norm regularization over the ℓ_1-norm regularization.

The committee in BAgg-IE consists of three classifiers[3], trained over disjoint random splits of the documents, which leads to different feature spaces for each, and with balanced labels (i.e., same number of useful and useless documents). Finally, to obtain the score of a text document we sum over the normalized scores of each classifier $s(\mathbf{d}) = \frac{1}{1+e^{-(\mathbf{w}^\top \mathbf{d}+b)}}$, which accounts for the differences in the feature weights of each classifier. In this equation, $\mathbf{w}$ and b are the weight vector and bias factor, respectively, of the classifier. Because BAgg-IE models the usefulness of a document independently of other documents, BAgg-IE is a pointwise learning-to-rank method.

[3]We experimented with additional classifiers, which slightly improved performance at the expense of substantial overhead.

In summary, BAgg-IE addresses the ranking problem as an optimized classification problem. In contrast, our second technique, RSVM-IE, which we describe next, adopts a principled learning-to-rank approach natively.

6.2.1.2 RSVM-IE: A Pairwise Ranking Approach

Our second learning-to-rank strategy, namely, *RSVM-IE*, is based on RankSVM [Joa03], a popular and effective pairwise learning-to-rank approach. Just as we did for BAgg-IE, we need to modify RankSVM's original optimization problem so that it incorporates in-training feature selection and, in turn, suits our ranking problem. In a nutshell, RankSVM scores the documents via a linear combination of the document features: the score of a document d is $s(d) = \sum_i w_i \cdot d_i$, where w_i is the weight of feature i and d_i is the value of feature i in document d. The objective of RankSVM is then to find the set of weights $\mathbf{w} = \{w_1, ..., w_n\}$ that is optimized to determine, in a pair of documents, if a document is more relevant than the other document. To achieve this, RankSVM learns the feature weights by comparing the features of useful and useless documents in pairs: each pair includes a useful and a useless document, and the label indicates whether the useful document is the first document in the pair.

By integrating the in-training feature selection discussed above into the original RankSVM formulation, we obtain the following optimization problem to solve for RSVM-IE:

$$\arg\min_{\mathbf{w}} \lambda_{All} \cdot \left(\frac{\lambda_{L2}}{2} \|\mathbf{w}\|_2 + (1 - \lambda_{L2}) \|\mathbf{w}\|_1 \right) + \sum_{(i,j)\epsilon\mathcal{P}} \ell(y \langle \mathbf{w}^\top, \mathbf{d}_i - \mathbf{d}_j \rangle)$$

where all variables are defined as for BAgg-IE, and $\mathbf{d}_i$ and $\mathbf{d}_j$ represent a useful and a useless document, respectively. For online learning, and in contrast to BAgg-IE, which uses the individual documents in the Pegasos scheme, the training examples are the pairs of useful and useless documents that the extraction process observes, which is known as *Stochastic Pairwise Descent* [Scu09].

Unlike BAgg-IE, RSVM-IE is designed from the ground up to address a ranking task, so we expect it to outperform BAgg-IE. Moreover, we expect the overhead of RSVM-IE to be substantially lower than that of BAgg-IE, since BAgg-IE maintains multiple learned

models (i.e., the classifiers in the committee). This overhead becomes noticeable when the models are frequently updated. Next, we explain our approach to decide when an update of the ranking models is desirable during the extraction process, thus reducing the overall document re-ranking overhead.

6.2.2 Update Detection

As we mentioned in Section 6.2.1, our adaptive extraction approach revises the ranking decisions periodically, to account for the new observations gathered along the extraction process. To determine when to update the ranking model (and, correspondingly, the document ranking), we introduce the *Update Detection* step (see Figure 6.2). To make this decision, we analyze whether the features of recently processed documents differ substantially from those in the ranking model. If this is the case, then we trigger a new ranking generation step (Section 6.2.1), which uses the recently processed documents as additional training examples. The new training examples often reveal novel features, or lead to adjusting the weight of known features, which in turn helps to more effectively prioritize the yet-unprocessed documents.

One possible approach for update detection is through feature shifting detection techniques [GLM12]. Feature shifting predicts whether the distribution of features in a (test) dataset differs from the distribution of the features in the training data. Unfortunately, most feature shifting techniques are problematic: First, they rely on computationally expensive algorithms (e.g., kernel-based one-class SVM classifier [GLM12]), thus incurring substantial overhead when applied repeatedly. Second, these techniques only detect changes in existing features, so they do not handle well the evolving feature space in our problem. Thus, the features that do not appear in the ranking would not be considered in the comparison, unless we re-train the kernel-based classifier from scratch, which would be prohibitively expensive.

As efficient alternatives, we introduce two update detection approaches, namely, Top-K and Mod-C. Top-K evaluates a reduced set of highly relevant features, determined independently from the ranking model, whereas Mod-C directly manipulates the low-level characteristics of the ranking model to detect changes in the feature space.

6.2.2.1 Top-K: Relevance-Based Update Detection Approach

Our first approach, namely, Top-K, exploits the fact that the predicted usefulness of the documents in the current ranking varies the most when the highly influential features in the ranking model change. For instance, if the word "lava" becomes more frequent along the processed useful documents in our *Occurs-in* example, this feature will become (temporarily) more relevant than others. In that case, the predicted usefulness of documents that include such word should increase accordingly to be prioritized over other documents. Based on this observation, Top-K compares the K most influential features in the current ranking against the K most influential features according to the recently processed documents, and triggers an update when the difference between these two sets exceeds a given threshold τ, determined experimentally, as we explain in Section 6.3. Overall, Top-K consists of two key steps: (i) *feature selection*, which selects the K most influential features; and (ii) *feature comparison*, which measures the distance between two sets of features. To perform feature selection, we choose the K features with highest weight in an SVM-based linear classifier trained—and subsequently updated—on the same features (i.e., words and tuple attributes) as the ranking algorithm. To perform feature comparison, we compute a generalized version of the Spearman's Footrule[4] [KV10], which considers the relative position of the features and their weights. According to this measure, the difference between feature weights will be higher when heavily weighted features change positions.

As discussed, Top-K maintains its own set of relevant features according to an SVM-based binary classifier. The advantage of this approach is that it makes Top-K independent of the ranking technique. However, the relevant features in this classifier may differ from those in the ranking model [HGO00]. In our *Occurs-in* example, for instance, a trained RankSVM model weighted the word "northern" as a top-20 feature, whereas a linear SVM model trained on the same documents weighted "northern" almost neutrally. Such discrepancies in the feature relevance may cause updates that have little impact on the document ranking or, alternatively, may lead to missing necessary updates because important features are not being evaluated. We now introduce Mod-C, which works directly with the ranking

[4]The generalized version of the Spearman's Footrule that we use is given by $\sum_i w_i \cdot \left|\sum_{j:j\leq i} w_j - \sum_{j:\sigma(j)\leq\sigma(i)} w_j\right|$, where $\sigma(i)$ is the rank of feature i and w_i is its weight.

models, to capture feature relevance directly.

6.2.2.2 Mod-C: A Model-Based Update Detection Approach

The techniques in Section 6.2.1 learn ranking models that consist of a vector of numeric weights, where each weight represents the captured relevance of one feature. We can then use a vector similarity metric, such as cosine similarity [MRS08], to measure the difference between the relevance of features in two similar ranking models. Our second technique, namely, *Mod-C*, exploits this observation and compares the current ranking model to an "updated" ranking model that also includes some of the recently processed documents. This updated ranking model includes only a fraction ρ of the recently processed documents, since including all of these documents would incur substantial overhead. To compare the ranking models, Mod-C depends on a metric suitable for the ranking model (e.g., cosine similarity for RSVM-IE) and a threshold α, determined experimentally as we explain in Section 6.3, that needs to be exceeded to trigger an update. In our cosine similarity example, α would indicate the maximum allowed angle between ranking models, hence triggering an update when this angle is exceeded. Mod-C is thus able to handle the real relevance of features, crucial to precisely decide when an update in the ranking model will improve the current document ranking.

In summary, we propose two update detection techniques that decide efficiently when it is beneficial to revise the ranking decisions to adaptively improve the extraction process.

6.3 Experimental Settings

We now describe the experimental settings for the evaluation of our adaptive ranking approach:

Collections: We used the *NYT Annotated Corpus* [San08], with 1.8 million New York Times articles from 1987 to 2007. We split this corpus into a training set (97,258 documents), a development set (671,457 documents), and a test set (1,086,944 documents). We evaluated different combinations of techniques and parameters on the development set. We ran the final experiments on the test set. Additionally, we used collections 1-5 from the

TREC conference [TRE00] to generate the queries for the query-based sample generation that we explain later in this section.

Document Access: As mentioned in Section 6.2.1, we consider two document-access scenarios: In the *full-access* scenario (see fully-accessible collection in Section 2.2.1), we rank all documents in a (moderately sized) document collection. In contrast, in the (more realistic) *deep-web* scenario (see deep web collection in Section 2.2.2), we retrieve the documents to rank through keyword queries. We evaluate our ranking approach over both scenarios. For the deep-web scenario we learn the queries following QXtract (Section 6.1) to retrieve an initial pool of documents. Also, we provide a search interface over our collection using the Lucene indexer [Luc15], to retrieve additional documents as the extraction process progresses: after each ranking update, we use the top-100 features of the updated ranking model as individual text queries to retrieve additional (potentially) useful documents.

Relations: Table 6.1 shows the broad range of relations from different domains that we extract for our experiments, with the number of useful documents for each relation in the test set. Our relations include *sparse* relations, for which a relatively small fraction of documents (i.e., less than 2% of the documents) are useful, as well as *dense* relations.

Information Extraction Techniques: We selected the extraction approach for each relation to include a variety of extraction approaches (i.e., both machine learning and rule-based approaches, as well as techniques with varying speed). Specifically, we considered different entity and relation extractors for each relation, and selected the best performing combination. However, for diversity, whenever we had ties in performance, we selected the (arguably) less common contender (e.g., a pattern-based approach to extract organizations and Maximum Entropy Markov Model [MFP00], or MEMM, for natural disasters):

- For the Person–Organization Affiliation relation we used Hidden Markov Models [EB07] and automatically generated patterns [WKPU08] as named entity recognizers for Person and Organization, respectively. We used SVM [GLR06] to extract the relation.

- For the Disease–Outbreak relation we used dictionaries and manually crafted regular expressions as named entity recognizers for Disease and Temporal Expression, respectively. We used the distance between entities to predict if they are related.

Relation	Useful Documents
Person–Organization Affiliation	185,237 (16.95%)
Disease–Outbreak	847 (0.08%)
Person–Career	458,294 (42.16%)
Natural Disaster–Location	18,370 (1.69%)
Man Made Disaster–Location	15,837 (1.46%)
Person–Charge	19,237 (1.77%)
Election–Winner	5,384 (0.50%)

Table 6.1: Relations for our experiments.

- For the remaining relations, we used Stanford NER [Sta15b] to find Person and Location entities, a MEMM [MFP00] to find Natural Disasters, and Conditional Random Fields [ML03] to find the remaining entities. Then, we used the Subsequence Kernel [BM05b] to identify relations between these entities.

Development Toolkits: We used the following off-the-shelf libraries: (i) Lingpipe [lin15], for rule-based named entity extraction; (ii) OpenNLP [ope15b], for word and sentence segmentation; (iii) E-txt2DB [Etx12] and Stanford NER, to train and execute named entity extractors based on machine learning; and (iv) REEL (see Chapter 3), to train relation extraction models.

Sampling Strategies: We compared two techniques to collect the initial document sample for our ranking techniques (Section 6.2.1):

- *Simple Random Sampling (SRS):* SRS picks 2,000 documents at random from the collection (only for the *full-access* scenario).

- *Cyclic Query Sampling (CQS):* CQS (see Cyclic in Section 4.2.3) iterates repeatedly over a list of queries and collects the unseen documents from the next K documents that each query retrieves until it collects 2,000 documents. We learned 5 lists of queries using sets of 10,000 random documents (5,000 useful and 5,000 useless) from the TREC collection by applying the SVM-based method in QXtract [AG03].

Ranking Generation Techniques: We evaluated our ranking generation techniques from Section 6.2.1. To obtain the best parameters for these techniques, we performed several

experiments over our development set, varying λ_{All} and λ_{L2}. The parameter values that we determined experimentally are as follows: for *BAgg-IE*, $\lambda_{All} = 0.5$ and $\lambda_{L2} = 0.99$; while for *RSVM-IE*, $\lambda_{All} = 0.1$ and $\lambda_{L2} = 0.99$. Setting $\lambda_{L2} = 0.99$ results in an ℓ_1-norm weight of $1 - \lambda_{L2} = 0.01$. This weight in turn results in models with 10 times fewer features—which are hence 10 times faster—than models that only use the ℓ_2-norm. Higher ℓ_1-norm weights would lead to lower-quality ranking models, as discussed in Section 6.2.

We also evaluated the following (strong) baselines:

- *FactCrawl (FC):* FC corresponds to our implementation of FactCrawl [BLNP11a], as described in Section 6.1.

- *Adaptive FactCrawl (A-FC):* We produced a new version of FC that re-ranks the documents. Specifically, to make FC more competitive with our adaptive ranking strategies, A-FC recomputes the quality of the queries, and re-ranks the documents with these new values after each document is processed. In addition, A-FC learns new queries and retrieves more documents before every re-ranking step.

(We evaluated other approaches, such as QXtract [AG03] and PRDualRank [FC11], but do not discuss them further because FactCrawl dominated the alternatives that we considered.)

Update Detection Techniques: We evaluated our update detection techniques from Section 6.2.2:

- *Top-K:* We set $K = 200$, which experimentally led to high coverage of the relevant features and small overhead in feature comparison. We set $\tau = \varepsilon \cdot K$, where ε indicates how much each feature can change without impacting the ranking. We experimented with several values of τ and finally picked $\tau = 0.5$ ($\varepsilon = 0.0025$).

- *Mod-C:* We evaluated several combinations of ρ and α: the best value for ρ is 0.1, while the best angle values for α are 5° and 30° for RSVM-IE and BAgg-IE, respectively.

We also compared against the following baselines:

- *Wind-F:* We implemented a naïve approach for update detection that updates the ranking model after processing a fixed number of documents. We experimented with

several values and observed no substantial differences. We report our results for updating 50 times along the extraction process, which leads to updates after 13,429 and 21,739 documents for the validation and test sets, respectively.

- *Feat-S:* We implemented an efficient version of feature shifting [GLM12] using an online one-class SVM based on Pegasos [SSSS07]. We used a Gaussian kernel with $\gamma = 0.01$ and $k = 6$, as suggested in [GLM12]. Finally, we triggered an update when the geometrical difference $F = 1 - S$ exceeded a threshold $\tau = 0.55$. Since the features of the documents after each update tend to fluctuate, we only run Feat-S after processing 700 new documents or more.

Executions: We executed each experiment five times with different samples (i.e., five different random samples and five different sets of initial sample queries), to account for the effect of randomness in the results, and report the average of these executions.

Evaluation Metrics: We use the following metrics:

- *Average Recall:* Average recall is the recall of the extraction process (i.e., the fraction of useful documents in the collection that have been processed) at different points during the extraction (e.g., after processing $x\%$ of the documents) and averaged over all executions of the same configuration.

- *Average Precision:* Average precision the mean of the precision values at every position of the ranking [TS06], averaged over all the executions of the same configuration.

- *R-Precision:* R-precision is the precision@K value when K is the total number of useful documents in the collection, averaged over all the executions of the same configuration.

- *Area Under the ROC curve (AUC):* AUC is the area under the curve of the true positive rate as a function of the false positive rate, averaged over all the executions of the same configuration.

- *CPU Time:* CPU time measures the time consumed for extracting and ranking the documents.

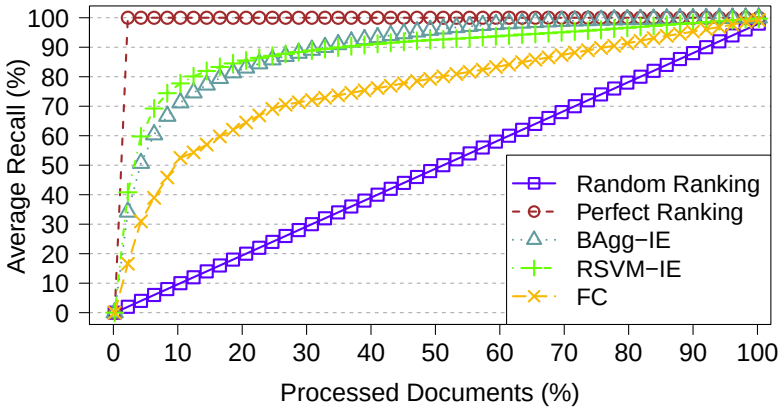


Figure 6.3: Average recall for Person–Charge for different base ranking generation techniques.

6.4 Experimental Results

We now present the results of the experimental evaluation of our adaptive ranking approach. We tuned the configuration of all components of our approach (i.e., the sampling strategy, the learning-to-rank approach, and the update detection approach) by exhaustively considering all possible combinations over the development set and selecting the best such combination. In the discussion below, for clarity, we consider the configuration choices for each component separately. Later, for the final evaluation of our approach over the test set and against the state-of-the-art ranking strategies, we use the best configuration according to the development set experiments.

6.4.1 Impact of Learning-To-Rank Approach

To understand the impact of using our learning-to-rank approach, we first evaluate our techniques of Section 6.2.1, *without the adaptation step*, against FC over the development set. Figure 6.3 shows the average recall for the Person–Charge relation for the full-access scenario. (For reference, we also show the performance of a random ordering of the documents, as well as of a perfect ordering where all useful documents are ahead of the useless ones.) Both RSVM-IE and BAgg-IE consistently outperform FC. Interestingly, RSVM-IE

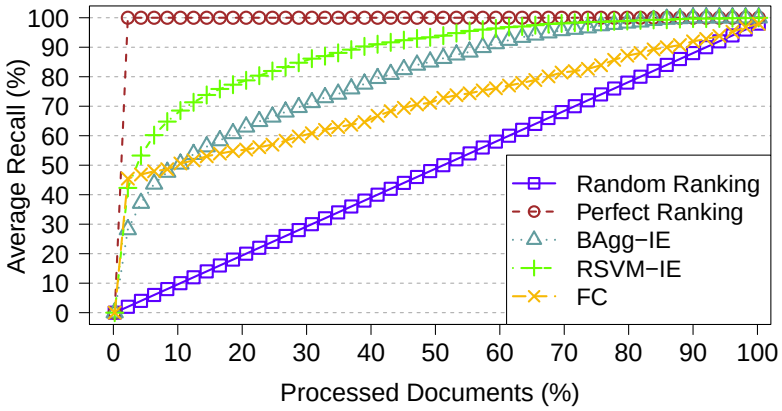


Figure 6.4: Average recall for Disease–Outbreak for different base ranking generation techniques.

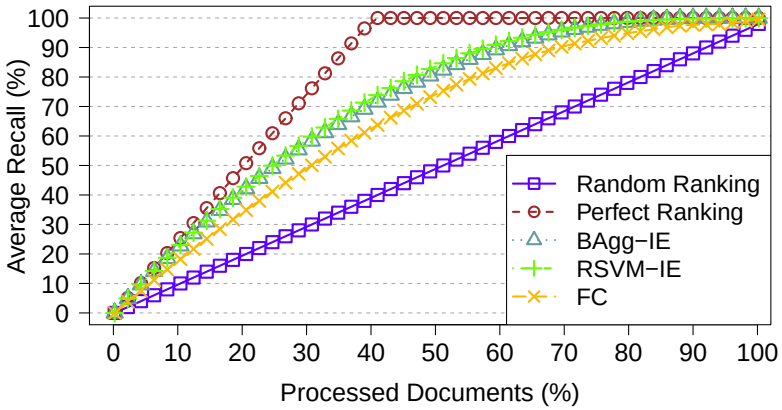


Figure 6.5: Average recall for Person–Career for different base ranking generation techniques.

performs better in early phases of the extraction, while BAgg-IE performs better in the later phases, which agrees with our intuition from Section 6.2.1: RSVM-IE is at its core a ranking optimization technique, while BAgg-IE is based on classifiers. BAgg-IE separates useful from useless documents, thus obtaining high-accuracy in the middle of the extraction process, which in turn leads to high recall later on. We observed similar results for most of the relations (e.g., Figures 6.4 and6.5 show the results for Disease–Outbreak and

Relation	Base SRS	Base CQS	Ada. SRS	Ada. CQS
Person–Organization Affil.	33.6±0.9%	37.9±1.0%	**44.2±0.3%**	43.6±0.3%
Disease–Outbreak	2.3±1.1%	3.1±0.6%	3.0±1.0%	**3.8±0.6%**
Person–Career	80.2±0.4%	79.2±0.5%	**84.2±0.2%**	84.1±0.2%
Natural Disaster–Loc.	6.1±1.1%	13.1±0.9%	10.2±0.9%	**16.4±0.8%**
Man Made Disaster–Loc.	7.3±1.8%	13.6±1.2%	12.9±1.4%	**17.2±0.8%**
Man Person–Charge	28.6±0.7%	28.1±1.1%	33.0±0.6%	**33.4±0.6%**
Man Election–Winner	6.6±4.0%	10.2±0.8%	9.4±3.2%	**12.6±0.6%**

Table 6.2: Average precision of different document sampling techniques on the ranking quality for all the relations with the base and adaptive versions of RSVM-IE for the full-access scenario.

Person–Career respectively). However, RSVM-IE performs better than BAgg-IE for sparse relations, so RSVM-IE is preferable for such relations even in later phases of the extraction process (see Figure 6.4). Overall, even without an adaptation step, our techniques outperform the state-of-the-art ranking technique FC.

6.4.2 Impact of Sampling Strategies

To understand the impact of different sampling techniques to learn the initial ranking model, we compared RSVM-IE and BAgg-IE using the SRS and CQS sampling techniques (Section 6.3). Figure 6.6 shows the average recall for the Man Made Disaster–Location relation in the full-access scenario for RSVM-IE, both without the adaptation step (denoted with keyword "Base" in the plot) as well as with adaptation (denoted with keyword "Adaptive"). (The results for BAgg-IE were analogous; see Figure 6.7.) Using CQS, a sophisticated sampling technique, has a generally positive impact relative to using the (simpler) SRS strategy. The only exceptions were the dense relations, namely, Person–Organization and Person–Career, for which a simple random sample typically includes a wide variety of useful documents, thus leading to high-quality models.

6.4.3 Impact of Adaptation

We claimed throughout this chapter that refining the document ranking along the extraction process significantly improves its efficiency. To support this claim, Figure 6.6 shows the average recall of RSVM-IE for the Man Made Disaster–Location relation for the full-access

Relation	Base SRS	Base CQS	Ada. SRS	Ada. CQS
Person–Organization Affil.	76.7±1.0%	77.7±0.9%	**82.7±0.1%**	**82.7±0.1%**
Disease–Outbreak	88.2±2.2%	87.9±0.9%	97.0±0.1%	**97.1±0.1%**
Person–Career	86.9±0.2%	86.5±0.4%	**89.9±0.1%**	**89.9±0.1%**
Natural Disaster–Loc.	64.0±3.5%	64.3±3.2%	**85.5±0.2%**	85.4±0.2%
Man Made Disaster–Loc.	67.4±3.2%	76.6±3.6%	88.6±0.2%	**89.2±0.1%**
Man Person–Charge	89.7±1.4%	87.3±1.6%	**95.5±0.0%**	95.4±0.0%
Man Election–Winner	79.5±8.6%	84.6±1.4%	94.9±0.5%	**95.3±0.1%**

Table 6.3: AUC of different document sampling techniques on the ranking quality for all the relations with the base and adaptive versions of RSVM-IE for the full-access scenario.

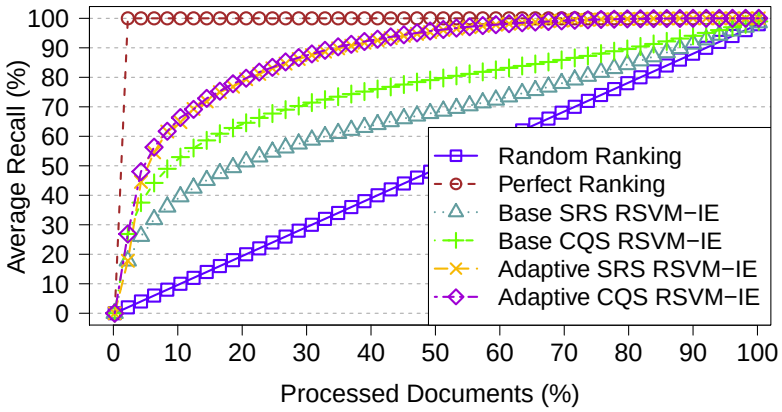


Figure 6.6: Average recall for Man Made Disaster–Location with different sampling techniques for the base and adaptive versions of RSVM-IE.

scenario. (The results for BAgg-IE are analogous, although the difference between the sampling techniques is higher than for RSVM-IE; see Figure 6.7.) These results show that by adapting the ranking model learned by RSVM-IE and, correspondingly, the document ranking, we significantly improve the efficiency of the extraction process. For example, Figure 6.6 shows that the adaptive versions of RSVM-IE can reach 70% of the useful documents after processing only 10% of the collection, whereas the base (non-adaptive) versions only reached 40% and 50% of the useful documents, for SRS and CQS, respectively. This same behavior was replicated by almost all relations. Additionally, as shown in Figure 6.6, the sampling technique does not have a significant impact anymore when we incorporate the

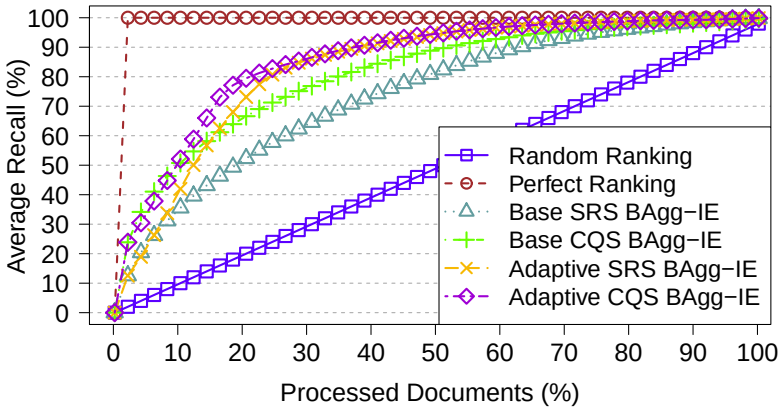


Figure 6.7: Average recall for Man Made Disaster–Location with different sampling techniques for the base and adaptive versions of BAgg-IE.

adaptation step. Nevertheless, we observed that the results of average precision and AUC (see Table 6.3) are generally better for CQS than for SRS, since CQS leads to processing more useful documents at early stages of the extraction process.

Finally, we evaluated the number of new features incorporated into the ranking model during the adaptation step. In early stages of the extraction process, an average of 200 (or about 25% of the total number of features in the previous models) are incorporated; a similar number of features is removed in each adaptation step. However, in later stages, this behavior changes as the models become more stable. Specifically, the number of incorporated and removed features drops to 10 after each adaptation step. These results show that while the initial adaptation steps significantly impact the ranking model, the later ones are insignificant. Therefore, it is important to properly schedule the adaptation step to avoid insignificant updates to the ranking model.

6.4.4 Impact of Update Detection

To evaluate the update detection techniques that we introduced in Section 6.2.2, we fix the document sampling to SRS, and evaluate the techniques according to their impact on the extraction process, distribution of updates, and overhead. Figure 6.8 shows the results of RSVM-IE for the Election–Winner relation for the full-access scenario. (The behavior

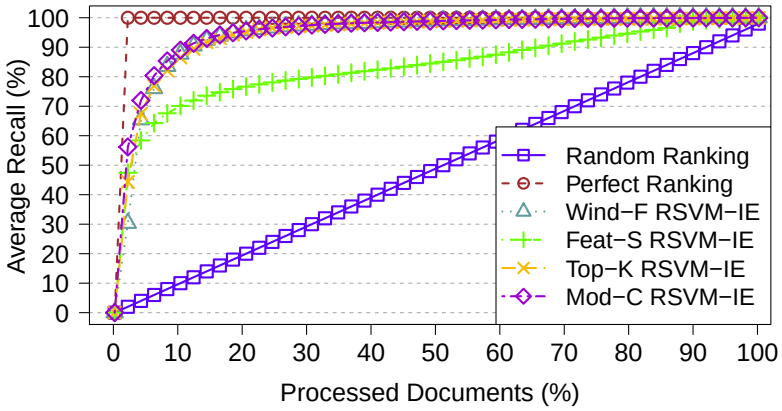


Figure 6.8: Average recall for Election–Winner for different update methods with RSVM-IE.

Relation	Wind-F	Feat-S	Top-K	Mod-C
Person–Organization Affil.	33.0±0.6%	29.7±1.1%	34.7±0.3%	**36.0±0.4%**
Disease–Outbreak	9.4±3.2%	11.8±1.5%	10.3±3.6%	**15.8±1.4%}**
Person–Career	84.2±0.2%	68.7±5.8%	**84.7±0.1%**	83.5±0.3%
Natural Disaster–Loc.	10.2±0.9%	8.4±0.7%	12.9±1.5%	**17.7±0.8%**
Man Made Disaster–Loc.	12.9±1.4%	13.3±1.7%	15.8±1.1%	**19.7±0.8%**
Person–Charge	33.0±0.6%	29.7±1.1%	34.7±0.3%	**36.0±0.4%**
Election–Winner	9.4±3.2%	11.8±1.5%	10.3±3.6%	**15.8±1.4%**

Table 6.4: Average precision of the update detection methods for the full-access scenario and using RSVM-IE as document ranking approach.

for the other relations is analogous. We show the average precision and R-precision of different update detection methods for all relations using the RSVM-IE approach in Tables 6.4 and 6.5). Feat-S technique performed poorly in comparison to others, because Feat-S stops performing updates when the features observed in the data stabilize with respect to its kernel-based definition of shifting. For this reason, Feat-S misses late updates that prioritize other still poorly ranked useful documents. In addition, we observe that both Top-K and Mod-C produce consistently better results than Wind-F, especially at early stages of the extraction process, thus leading to high recall early in the extraction process. Overall, we show that both Top-K and Mod-C are robust alternatives for update detection in terms of ranking quality.

Additionally, to evaluate the impact on efficiency of the update detection techniques, we

Relation	Wind-F	Feat-S	Top-K	Mod-C
Person–Organization Affil.	35.8±1.0%	37.3±0.8%	40.3±1.5%	**41.3±0.5%**
Disease–Outbreak	12.6±6.7%	23.0±2.6%	12.5±6.3%	**23.9±1.5%**
Person–Career	76.3±0.1%	64.1±5.0%	**76.6±0.0%**	75.1±0.5%
Natural Disaster–Loc.	12.0±1.7%	16.8±1.3%	22.1±1.4%	**25.0±0.2%**
Man Made Disaster–Loc.	14.6±3.3%	23.6±1.9%	25.3±2.3%	**28.9±0.4%**
Person–Charge	35.8±1.0%	37.3±0.8%	40.3±1.5%	**41.3±0.5%**
Election–Winner	12.6±6.7%	23.0±2.6%	12.5±6.3%	**23.9±1.5%**

Table 6.5: R-precision of the update detection methods for the full-access scenario and using RSVM-IE as document ranking approach.

Update Technique	CPU Time per Document
Wind-F	**0.01±0.00 ms**
Feat-S	5.72±0.29 ms
Top-K	1.89±0.71 ms
Mod-C	0.32±0.10 ms

Table 6.6: Average CPU time to perform update detection.

calculated the overhead per document in terms of average CPU time, which we summarize in Table 6.6. As expected, Wind-F incurs negligible overhead (roughly 0.01 ms per document), since it only keeps a counter of the processed documents, whereas Feat-S incurs the highest overhead (5.72 ms per document). Our two techniques, Top-K (1.89 ms per document) and Mod-C (0.32 ms per document), exhibit a substantial difference in terms of efficiency, since the overhead of Top-K is dominated by the use of the binary classifier, as we discussed in Section 6.2.2.

We also studied the distribution of updates across the extraction process, to understand the behavior of Top-K and Mod-C. Figure 6.9 shows the number of updates that each technique performs at different stages of the extraction process. Top-K and Mod-C tend to update much more frequently in early stages, where almost all documents carry new evidence of usefulness, than in later stages. For instance, most of the updates are performed while processing the first 10% of the collection. This behavior leads to ranking models that stabilize early, since they are able to overcome the usual lack of training data in the initial document samples. Interestingly, despite the density of updates early in the process, the overall number of updates of Top-K and Mod-C remains smaller than that of Wind-F, since

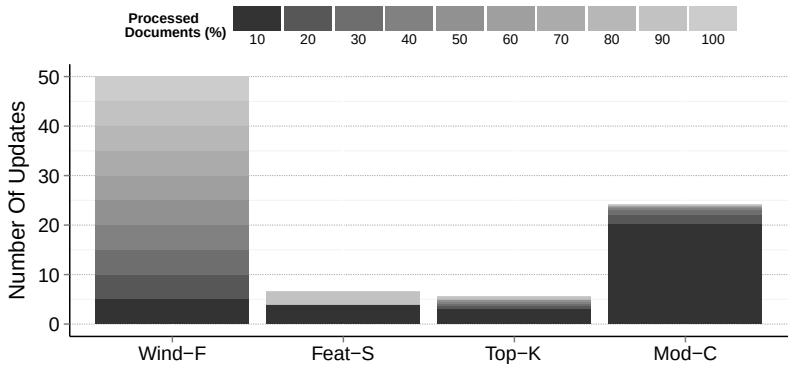


Figure 6.9: Distribution of updates for different techniques over the Election–Winner relation with RSVM-IE. (Darker shades represent earlier stages of the extraction process.)

our techniques avoid unnecessary updates in late phases of the extraction process.

Additionally, we studied the number and percentage of features that are incorporated to, removed from, and updated in the models. Figure 6.10 shows these values together with the total number of features in the model for Mod-C, the best performing update detection method based on our analysis above, and Wind-F, which updates the models at regular intervals: The adaptation steps triggered by Mod-C add a consistent number of new features (i.e., about 100 per adaptation step) throughout the extraction process. (Because of the regularization step during learning, the model removes a comparable number of features from the model.) This behavior significantly differs from that of Wind-F, which incorporates a large fraction of new features in early phases of the extraction process but only a small number of features later on: Mod-C (and also Top-K) only performs an update if this update will have a significantly positive impact on the model. This is reflected in the number of features that change in the model after each update (see Updated line in Figure 6.10), which for Mod-C are consistently close to the total number of features (see All line in Figure 6.10). Finally, relations such as Disease–Outbreak, which include(very) few useful documents in the initial document sample, tend to incorporate more features in the ranking model. This occurs because most useful documents processed along the extraction process will include unseen features. In conclusion, and considering also the

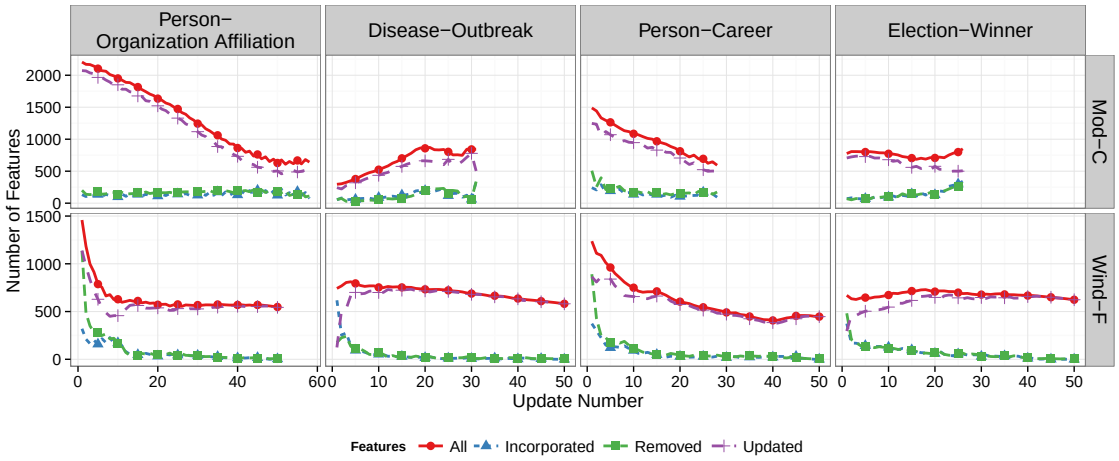


Figure 6.10: Analysis of the feature space during updates for RSVM-IE and a selection of relations and update detection methods.

efficiency results, Mod-*C* consistently outperforms the other techniques.

6.4.5 Scalability of our Approach

To understand how our strategies scale with the document collection size, we produced 10 subsets of the test collection with different sizes (from 10% to 100% of the total collection) and we measured (i) the time overhead for producing the ranking and (ii) the extraction time needed to reach a (fixed) target number of useful documents in each subset. Figure 6.11 shows how the size of the collection affects the CPU time needed to perform the ranking and extraction tasks with our techniques for the Natural Disaster–Location relation: the CPU time needed to perform an extraction task with our techniques grows approximately linearly with the collection size, which is desirable. Additionally, Figure 6.12 shows—for the Person–Organization Affiliation relation—that the time needed to find and process a target number of useful documents significantly drops as we increase the size of the collection. In this figure, the target number of useful documents corresponds to that in the subset of the collection that only contains 10% of the documents. As shown, the time becomes almost constant when the number of useful documents in the subset is large enough for the ranking to reach the target number at very early phases of the extraction process.

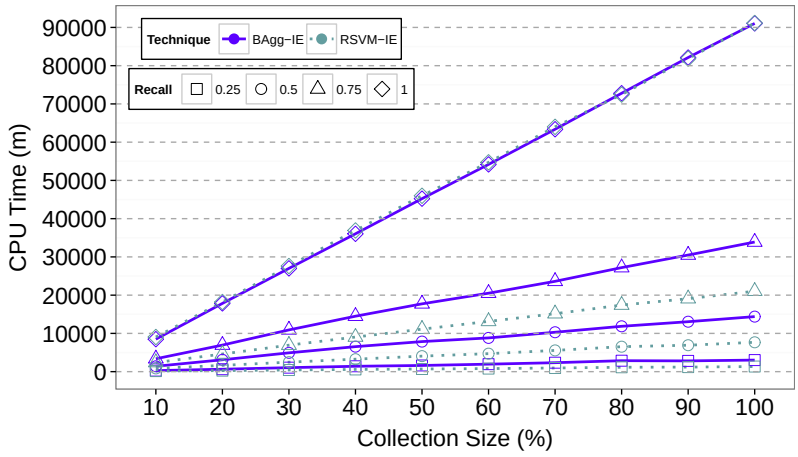


Figure 6.11: Average CPU time of our techniques as a function of the collection size for different target recall values, for the Natural Disaster–Location relation.

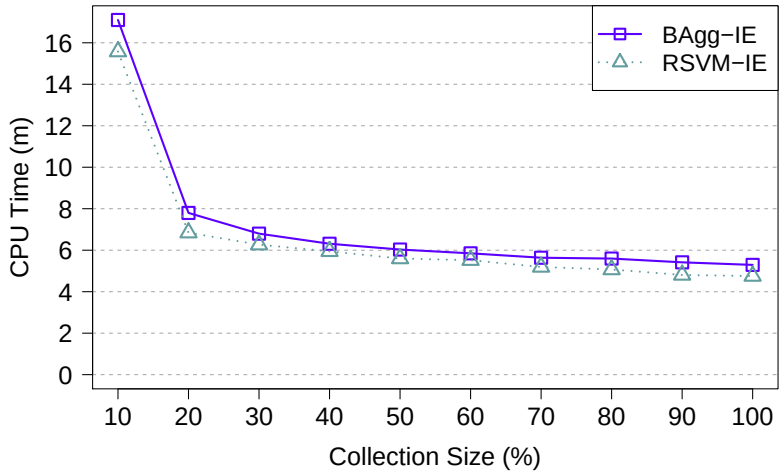


Figure 6.12: Average CPU time to find a target number of documents (i.e., the number of useful documents in the subset with 10% of the collection) for the Person–Organization Affiliation relation, as a function of the collection size.

6.4.6 Comparison with State-of-the-Art Ranking Strategies

We now compare our best performing ranking approaches with the state-of-the-art approaches discussed in Section 6.1. We selected the best configuration for RSVM-IE and BAgg-IE according to the previous experiments, which involve CQS sampling and Mod-C update detection. Then, we ran this configuration over the test set to compare with FC and A-FC. We compare the techniques on ranking quality and efficiency.

Relation	BAgg-IE	RSVM-IE	FC	A-FC
Person–Organization Affil.	40.5±0.9%	**45.7±0.3%**	29.0±0.9%	30.5±0.6%
Disease–Outbreak	3.5±1.3%	**8.3±0.2%**	1.5±0.4%	1.6±0.4%
Person–Career	79.2±0.4%	**85.1±0.1%**	66.3±1.1%	63.2±1.0%
Natural Disaster–Loc.	10.2±1.4%	**18.9±0.6%**	6.0±0.4%	7.1±0.4%
Man Made Disaster–Loc.	10.8±2.1%	**17.0±0.1%**	3.8±0.4%	4.1±0.4%
Person–Charge	22.3±2.6%	**33.8±0.3%**	10.0±1.5%	11.0±1.2%
Election–Winner	9.6±0.6%	**15.5±0.3%**	2.4±0.2%	2.6±0.2%

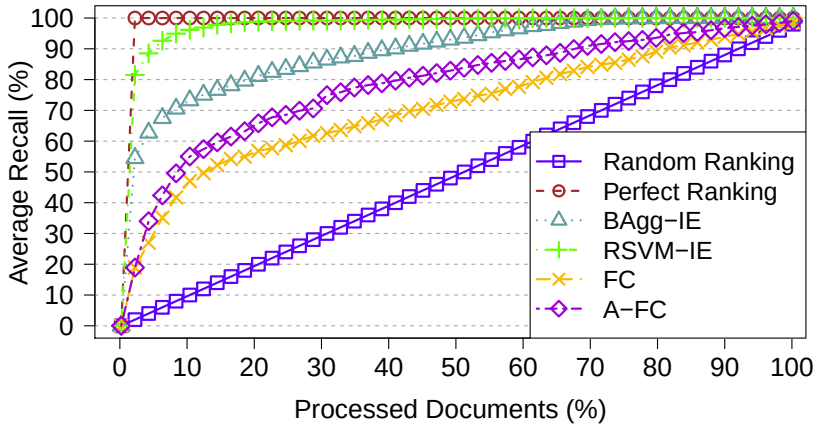
Table 6.7: Average precision of the rankings generated by different techniques for the full-access scenario.

Relation	BAgg-IE	RSVM-IE	FC	A-FC
Person–Organization Affil.	78.2±0.6%	**82.4±0.1%**	68.9±0.5%	71.9±0.8%
Disease–Outbreak	89.7±0.3%	**98.2±0.1%**	71.5±11.4%	78.8±5.4%
Person–Career	83.7±0.4%	**88.6±0.1%**	76.3±0.4%	72.9±0.5%
Natural Disaster–Loc.	78.4±0.5%	**85.8±0.1%**	67.8±1.5%	72.9±0.2%
Man Made Disaster–Loc.	81.4±1.2%	**88.0±0.0%**	67.1±1.7%	69.9±1.5%
Person–Charge	90.5±2.1%	**95.1±0.0%**	74.6±2.8%	78.9±1.5%
Election–Winner	90.2±0.2%	**95.4±0.1%**	78.1±1.5%	80.5±1.3%

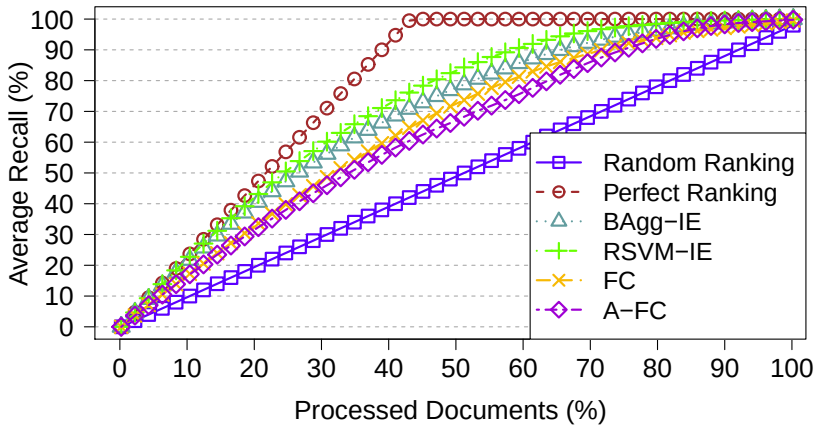
Table 6.8: AUC of the rankings generated by different techniques for the full-access scenario.

Relation	BAgg-IE	RSVM-IE	FC	A-FC
Person–Organization Affil.	40.6±0.9%	**46.8±0.1%**	29.1±0.9%	31.1±1.0%
Disease–Outbreak	3.5±0.9%	**7.6±0.2%**	1.5±0.4%	1.6±0.4%
Person–Career	79.3±0.6%	**85.9±0.0%**	66.3±1.1%	63.2±1.0%
Natural Disaster–Loc.	10.3±1.6%	**18.9±0.6%**	6.0±0.4%	7.1±0.3%
Man Made Disaster–Loc.	10.4±1.7%	**17.3±0.2%**	3.8±0.4%	4.1±0.4%
Person–Charge	21.9±2.5%	**34.2±0.4%**	10.0±1.5%	11.0±1.3%
Election–Winner	9.9±0.3%	**15.6±0.3%**	2.4±0.2%	2.6±0.2%

Table 6.9: Average precision of the rankings generated by different techniques for the deep-web scenario.



(a) Disease–Outbreak

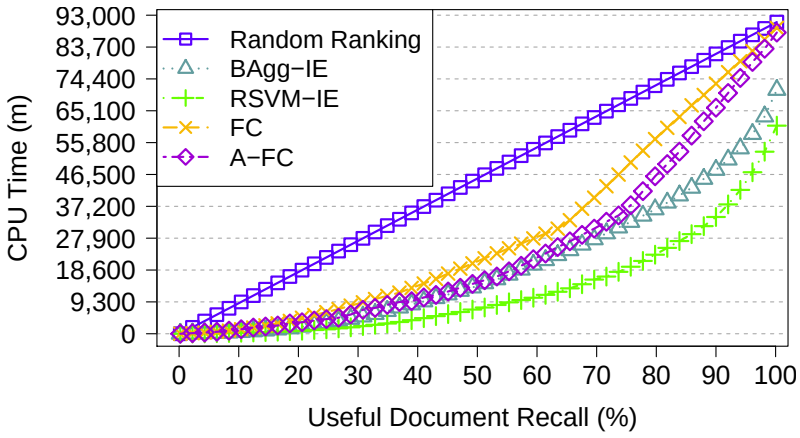


(b) Person–Career

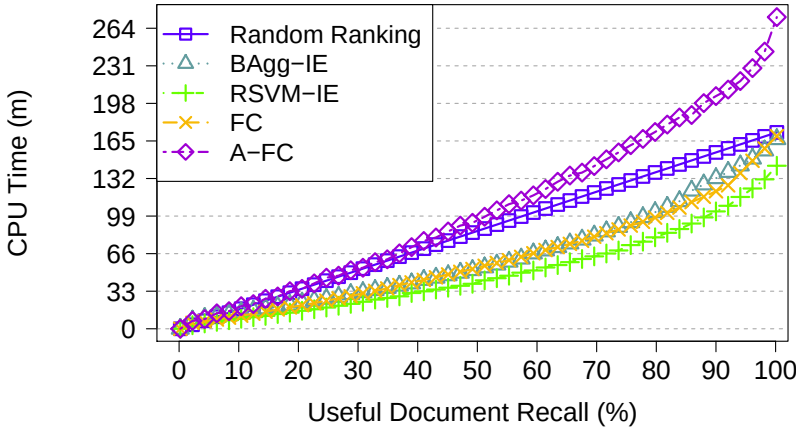
Figure 6.13: Average recall for different ranking approaches in the full-access scenario.

Table 6.8 shows the average precision and AUC of the four techniques that we compare, for all relations and over the full-access scenario: RSVM-IE and BAgg-IE generally outperform the FactCrawl baselines by a large margin, and RSVM-IE consistently outperforms BAgg-IE. We performed this experiment in the deep-web scenario as well, with similar conclusions (see average precision in Table 6.9). Interestingly, our adaptive version of FactCrawl, A-FC, does not exhibit the same significant improvement compared to FC that we observed between the adaptive and base versions of RSVM-IE and BAgg-IE above: A-FC is unable to properly model the usefulness of the documents when new features emerge, since it only relies on a small number of features.

To understand the effects of the relation characteristics, we studied the performance



(a) Natural Disaster–Location



(b) Person–Organization Affiliation

Figure 6.14: CPU time to obtain a target recall value.

of the techniques over both sparse (Figure 6.13a) and dense (Figure 6.13b) relations. The performance gap is more evident for sparse relations than it is for dense relations: The vocabulary around mentions of sparse relations tends to be reduced and specific, which makes it easier to model and prioritize the useful documents. Conversely, dense relations are scattered across diverse documents, thus co-occurring with a large variety of words, which makes it difficult to select a set of features that precisely identifies useful documents. Regardless, RSVM-IE and BAgg-IE still outperform the other techniques, since they are able to handle feature spaces of variable sizes.

We evaluate efficiency by measuring the time—including both ranking and extraction time—that each technique requires to achieve different values of recall. We show the results

for two relations that exhibit substantially different extraction times according to their respective information extraction system: (i) Natural Disaster–Location, which takes an average of 6 seconds per document (Figure 6.14a); and (ii) Person–Organization Affiliation, which takes an average of 0.01 seconds per document (Figure 6.14b). RSVM-IE outperforms the others, in agreement with our earlier findings. The results for Person–Organization Affiliation are, in contrast, slightly different. For this fast extraction task, the overhead of the ranking technique can be problematic since it may easily become larger than the extraction time per se. We can observe such behavior for A-FC, which is less efficient than a random ranking technique with no overhead: A-FC (and, correspondingly, FC) relies on features that are expensive to compute [BLNP11a], which is problematic for the adaptive case. However, the other techniques behave similarly as for the more expensive relations, with RSVM-IE resulting in the most efficient extraction process. Interestingly, for extraction tasks that incur lengthier extraction time, as is the case for Natural Disaster–Location, the quality of the ranking has a higher impact on efficiency than for other extraction tasks.

Overall, our experiments show that RSVM-IE outperforms all other techniques in all settings and extraction tasks. More specifically, RSVM-IE produces better rankings, while incurring very little overhead. Finally, when combined with Mod-C, RSVM-IE achieves much lower extraction times than the alternative strategies that we studied. Indeed, even with fast information extraction systems, adaptively ranking documents with RSVM-IE remained the best choice. Additionally, we evaluated the scalability of our techniques and confirmed that as the size of the collection grows, so does the positive impact of our approach, making it a substantial step towards scalable information extraction.

6.5 Conclusions

In this chapter, we presented an adaptive, lightweight document ranking approach for information extraction. Our approach enables effective and efficient information extraction over large document collections. Specifically, our approach relies on learning-to-rank techniques that learn in a principled way the fine-grained characteristics of the useful documents for an extraction task of interest. Our techniques incorporate (i) online learning algorithms,

to enable a principled, efficient, and continuous incorporation of new relevant evidence as the extraction process progresses and reveals the real usefulness of documents; and (ii) in--training feature selection, to enable the learning of ranking models that rely on a small, discriminative set of features. Our experiments show that our approach exhibits higher recall and precision than state-of-the-art approaches, while keeping the overhead low.

The key contributions of this chapter are the techniques that we have devised for the two critical building blocks of our approach, namely, ranking generation and update detection. Beyond being efficient and effective, which is crucial for the performance of the overall extraction process, these techniques are also remarkably flexible: These techniques can potentially handle different, dynamic sets of features without further modifications. We believe that these techniques will enable the development and deployment of highly effective, domain-specific document ranking approaches. Overall, our document ranking approach and techniques are a substantial step towards scalable information extraction.

Chapter 7

Ranking Sentences for Scalable Information Extraction

In Chapter 6, we addressed the problem of document ranking for efficient information extraction, namely, to prioritize the extraction effort over useful documents for an extraction task of choice. We also showed that our techniques need to process considerably fewer documents than state-of-the-art approaches to extract tuples from the same number of useful documents. However, information extraction systems typically operate over small text units (e.g., sentences or paragraphs) rather than over the entire documents, as discussed in Chapter 2. Even more importantly, many times only a small number of sentences in a useful document are useful for the extraction task at hand [NVB01]. We argue that further efficiency improvements can be made by focusing on these few useful sentences and ignoring the rest.

In this chapter, we present a sentence ranking approach to effectively and efficiently prioritize the useful sentences that lead to the extraction of novel, unseen tuples for an extraction task of interest. Specifically, we propose a principled, efficient approach that exploits a forward greedy sparse group selection strategy [LSA09] to identify the (often-few) useful sentences from a set of documents. Our approach models each sentence as a group of n-grams and iteratively selects the sentence that best explains a carefully designed representation of the extraction task at hand derived from discriminative n-grams for the

task. Importantly, we build this representation of the extraction task gradually, as the extraction process progresses, to capture all relevant aspects of the task. During sentence ranking, our approach updates this representation to account for the relevant aspects of the extraction task that have been already explained by other previously selected sentences. By doing this, our approach manages to prioritize sentences that lead to the extraction of unseen tuples. We exploit distributed word representations to model the sentences and the extraction task, to enable a meaningful characterization of sentence usefulness and novelty. Our experimental evaluation over a broad range of extraction tasks shows the merits and limitations of all relevant building blocks in our approach and, more importantly, shows the significant efficiency improvements that can be made by effectively prioritizing sentences.

In summary, the main contributions of this chapter are:

- An end-to-end sentence ranking approach for effective and efficient information extraction in a greedy, efficient, and principled manner (Section 7.2). Our approach produces rich, compact representations of both the sentences and the extraction task, to effectively characterize usefulness and novelty (Section 7.2.3). Also, our approach prioritizes sentences that lead to the extraction of unseen, novel tuples by exploiting sparse group selection solutions over the sentences and with respect to the extraction task (Section 7.2.4). Finally, our approach enables trading relevance and novelty, which largely benefits the requirements of diverse downstream applications (Section 7.2.5).

- An experimental evaluation of our approach using multiple extraction tasks implemented with a variety of extraction approaches (Sections 7.3 and 7.4). Our evaluation highlights the substantial efficiency improvements—in terms of processing time—that can be achieved by effectively prioritizing sentences. In particular, we show the merits and limitations of all relevant building blocks in our approach for identifying novel sentences for an extraction task of interest.

We now review necessary background and define our problem of focus in this chapter.

7.1 Background and Problem Definition

To extract tuples from a given text document, information extraction systems often run over all sentences in the document, as described in Chapter 2. However, such exhaustive processing is many times unnecessary even over a useful document, because only a small fraction of the sentences are useful for the extraction task at hand [NVB01].[1] For our *Occurs-in* relation, for instance, under 4% of the sentences in useful documents in TREC 1-5 collections [TRE00] are useful when processed with a state-of-the-art information extraction system. If we could identify the small fraction of useful sentences, we would complete the extraction process in substantially less time than an exhaustive execution and without any need to change the information extraction system.

Early approaches for identifying useful sentences for an extraction task of interest have resorted to filtering [Agi05]: In a preprocessing step, these approaches run inexpensive text classifiers [NVB01; BHL11] or hand-crafted patterns [GHY02] over each sentence to decide whether the sentence should be forwarded to the extraction system (e.g., if the classifier categorizes the sentence as positive or if the pattern matches the sentence). By applying these filtering techniques the extraction process becomes noticeably more efficient because the extraction techniques will run over a (relatively) small number of sentences [WSE13].[2] Unfortunately, filtering approaches have two crucial limitations. First, recall of the extraction process (i.e., the number of tuples that are extracted with the information extraction system) may suffer, because filtering approaches are usually far from perfect [PR07]. As a result, many useful sentences are not forwarded to the information extraction system. Wachsmuth et al. [WSE13] propose trading recall for efficiency by increasing the scope of the filtering approach (e.g., to paragraphs or to the entire document); however, recall of the extraction process still remains affected. Second, efficiency may also be compromised, because the information extraction system may run over sentences that produce already

[1] Our approach is not applicable over open information extraction scenarios (e.g., [BCS+07]) where sentences frequently contribute tuples to the open-ended extraction task.

[2] The extraction process may also become more effective, as demonstrated empirically in [PR07]. This happens because reducing the number of sentences to process with the information extraction system reduces, in effect, the number of false positives (i.e., extracted tuples that are semantically incorrect). We do not consider effectiveness in our work. Instead, we trust the output of the information extraction system and focus on efficiently and effectively identifying useful sentences for our extraction task of interest.

seen tuples that may be ignored by downstream applications.

To alleviate the limitations above in the (related) task of efficiently extracting from text attributes (e.g., *work place* or *location*) of entities (e.g., *person* or *organization*), Xu et al. move beyond filtering and adopt instead a ranking approach [XGZ11]. Following an idea similar in spirit to those of QXtract [AG03] and FactCrawl [BLNP11b], their approach collects a set of passages (e.g., sentences, paragraphs, or fixed- or variable-size text windows) that are likely to be useful for the extraction task of choice. Specifically, they issue automatically learned discriminative queries for the task to a passage retrieval system, namely, a retrieval system that identifies passages—from indexed documents—that are topically relevant to a given query [KZ01]. This approach then re-ranks the retrieved passages according to a passage score that considers: (1) occurrence of the entity of interest (e.g., *Reed Hastings*) or, alternatively, of pronouns that can potentially refer to the entity; (2) novel or repeated occurrence of an entity of the attribute type to be extracted (e.g., *work place*), where the novelty is defined according to previously processed passages; and (3) mention of any top-n most discriminative words, weighted proportionally to their precision (i.e., ratio of useful passages that include the word to the total set of passages that include the word) over a set of training passages.

Unfortunately, the approach above has some serious limitations. First, recall of the extraction process is still compromised because, just as in QXtract and FactCrawl, the learned queries may miss passages with mentions of the extraction task at hand. Second, its efficiency is also affected, since computationally expensive text processing steps are needed over all passages during scoring (e.g., named entity recognition needs to be performed in Item (2) above). Third, and similarly to QXtract and FactCrawl, this approach does not benefit from or adapt to the information that is captured as the extraction process progresses. Finally, novelty of the extraction output is considered in isolation, ignoring signals from the full contents of the passages that, as we will see, are helpful to characterize the novelty of the extraction output. This impacts the *diversity* of the extraction output, namely, the unique tuples and attributes that are extracted from the text, which is many times a desirable property of the output, as we will see.

Novelty and diversity have been extensively studied in the related area of information

retrieval, where the relevance of a document must be determined with respect to those appearing before it in query results [Gof64; Boy82]. More specifically, Clarke et al. distinguish novelty—the need to avoid redundancy in the returned results—and diversity—the need to resolve ambiguity in the posed search query [CKC+08]. Some approaches (e.g., [CG98; ZCL03; SMK09; AGHI09; DC12]) greedily apply carefully designed scoring functions to the documents to determine their novelty. Other methods (e.g., [CK06]) exploit the so-called negative relevance feedback, namely, to deem seen documents as non-relevant, to then resort to traditional relevance measures. Finally, learning-based approaches (e.g., [YJ08; SRG10; SMO10; RSJ12]) train ranking algorithms for diversity in the results from the ground up. Importantly, novelty and diversity of the output are highly desirable properties in multiple domains [SCAC14].

Unlike in information retrieval, though, our information extraction setting requires that we identify *novel sentences*, or (useful) sentences that lead to the extraction of unseen tuples, rather than documents that cover different topical aspects of a given query. As a result, we should effectively determine the novelty of a sentence—in terms of its (future) extraction output—with respect to previously processed sentences. Accordingly, by promoting novelty we enhance diversity over time, just as in the (related) area of recommender systems [VC11]. This is a particularly challenging proposition for three main reasons. First, sentences are generally short, which complicates effectively characterizing their novelty and usefulness. Our methods should therefore exploit this scarce information competently. Second, novelty of a sentence should be determined in a lightweight manner, for efficiency. In particular, determining the novelty of a sentence should avoid incurring extraction effort. Finally, the fraction of useful sentences for an information extraction task in a set of documents can many times be very small, as discussed. Our methods should accurately identify these few useful sentences from the documents.

Based on the discussion above, we now present our problem of focus in this chapter:[3]

Problem Definition 4 *Consider a set of text documents D and an information extraction*

[3] A more general version of this problem would be to prioritize text fragments of different lengths (e.g., a set of n sentences or paragraphs). We focus on sentences in our work but, as we will see, our techniques are applicable to other text units as well.

system E trained to extract tuples for a relation R from text. Our goal is to prioritize the extraction effort of E over the sentences S from the documents in D, so that: (i) we process the useful sentences for E earlier in the extraction process, for efficiency; and (ii) we extract tuples that are novel with respect to those from previously processed sentences, for diversity. Importantly, we want our ranking of sentences to achieve these goals while satisfying certain efficiency requirements (e.g., that running E over S in ranked order leads to a larger and more diverse set of tuples faster than running E over S directly). Moreover, our sentence ranking will have to adapt in light of the relevant information about the extraction task (e.g., the real usefulness of the sentences and their extracted tuples) obtained as E runs over sentences from S.

Because of the sparsity of useful sentences discussed earlier, the problem above has a counterpart in sparse group selection [HZM09]. In many high-dimensional learning problems, certain "parts" of the the data may be more relevant than others to the learning task. For example, the most relevant signals to categorize a text document in a text classification problem may lie in a few of its sentences rather than in the entire document [YS14]. This observation holds as well for many other text-centric problems such as sentiment analysis [YYC10; TM11] or citation prediction [YS14], to name a few. The key idea in sparse group selection is that variables in the same group (e.g., words in the same sentence) will be either deemed as relevant or irrelevant for the task simultaneously. We now describe our sentence ranking approach, which builds on an efficient forward greedy approach for the sparse group selection problem.

7.2 Ranking Sentences: A Group OMP-Based Approach

In Section 7.1, we argued that existing sentence filtering and ranking approaches for efficient information extraction have crucial limitations. We also argued that the problem of identifying useful sentences for an extraction task of choice has a counterpart in the sparse group selection problem. We now propose a sentence ranking approach to prioritize novel sentences for an information extraction task that is based on Group OMP [LSA09; LSA11], an efficient greedy strategy for the sparse group selection problem. Unlike tradi-

tional sentence filtering and ranking techniques, our approach improves the ranking decisions periodically in a robust manner, for novelty and efficiency. We first provide the necessary background on sparse group selection (Section 7.2.1). Then, we present an overview of our general approach (Section 7.2.2). Finally, we provide the details of our approach (Sections 7.2.3 through 7.2.5) and analyze its efficiency (Section 7.2.6).

7.2.1 Sparse Group Selection: Background

In tasks involving high-dimensional spaces, sparse group selection aims to identify a small number of groups of variables that collectively explain all aspects of a task sufficiently accurately. For example, assume a traditional high-dimensional learning task (e.g., text classification) defined in terms of $y \in \mathbb{R}^d$ (e.g., a vector of $\{+1, -1\}$ labels for d documents) and $X \in \mathbb{R}^{d \times p}$ (e.g., a set of d documents modeled with p-sized word feature vectors). Sparse group selection for this task aims to find a small set of groups features (e.g., a relatively small set of sentences) that best explain the vector y of observations, given certain group constraints defined over the p features (e.g., all words in a sentence form a group). This problem is generally approached as finding a vector $\beta \in \mathbb{R}^p$ of "relevant" coefficients for the features such that $||y - X\beta||_2^2$ is minimized, by imposing group sparsity constraints (i.e., that only a few coefficients—and from features of a few groups—in β are non-zero).

Notable solutions to the sparse group selection problem include Group Lasso and Group LARS [YL06], Elastic-Net [ZH05], and Group OMP [LSA09; LSA11]. In particular, Group Lasso has been the most extensively studied (e.g., see [KKK06; MVDGB08; RF08]) due to its simplicity: (i) In an intra-group level, Group Lasso "forces" the variables within a group to be comparably relevant via ℓ_2-norm, while (ii) in an inter-group level, it "forces" sparsity across the groups via ℓ_1-norm. Unfortunately, Group Lasso is problematic for our task for two main reasons. First, assessing the merits of a group with respect to the task, which is necessary for our task as we will see, is a difficult proposition in Group Lasso.[4] Second, interpreting the optimization decisions behind the selection of groups, so that we can characterize novelty and usefulness of sentences, is also difficult.

[4]Yang et al. [YXKL10] propose an online learning approach for Group Lasso that learns from one group at a time. Even in this formulation interpreting the impact of each group in the learning task is cumbersome, because the optimization decisions are performed globally.

A better suited approach for our problem is Group OMP [LSA09; LSA11], a variant of the forward greedy Orthogonal Matching Pursuit (OMP) algorithm [PRK93; DMA97] for sparse group selection. For a given task, Group OMP addresses the sparse group selection problem based on two main ideas: (i) to select a group of variables, Group OMP evaluates the correlation of each group to the current version of the task; and (ii) to keep the number of selected groups small, Group OMP updates the task in light of all previously selected groups so that only unexplained aspects of the task are considered. (We provide a more formal definition of Group OMP in Section 7.2.4.) By doing this, Group OMP approximates group-sparsity constraints to the number of non-zero coefficients (i.e., the ℓ_0-norm). As we see next, we can largely benefit from such (greedy) approach in our information extraction setting.

7.2.2 Overview of Our Approach

As discussed above, we rely on sparse group selection techniques for efficiently ranking novel sentences. The key intuition is that sentences that are relevant to a particular extraction task and that explain different aspects of the extraction task are likely to be novel, in that they are likely to lead to the extraction of unseen tuples. The following example illustrates the intuition behind our approach:

Example 3 *Assume a comprehensive and accurate representation U of the useful information for an information extraction task T. One possible U for our Occurs-in relation, for instance, consists of all words or phrases (e.g., "earthquake", "aftermath") positively correlated to natural disasters. Assume as well a set of documents D from which we want to efficiently extract tuples for T. To prioritize the extraction effort over sentences in D, we could measure the similarity of each sentence s_i to U (e.g., using cosine similarity over the words in the sentence) to, in turn, process the sentences in descending similarity order. Say that by processing the sentences in order we process a sentence s_U that produces a tuple t for T (i.e., s_U is useful for T). If we continue to process sentences in this order, our extraction system may run over another sentence s_D that leads to the extraction of the (seen) tuple t, which is many times undesirable. For example, an extraction system for our Occurs-in relation may extract tuple ⟨tsunami, Hawaii⟩ from sentence "A tsunami swept*

the coast of Hawaii" as well as from sentence "The coast of Hawaii was devastated after the tsunami." To prevent this, we could remove from U the aspects of T already explained by s_U, to obtain $\tilde{U}$, an updated version of U. For instance, we could remove all words and phrases in s_U from U.[5] *We expect the similarity between s_D and $\tilde{U}$ to be lower than over the original U. More importantly, we expect the similarity between sentences that differ from s_U—and are hence likely to produce different tuples—to be higher with respect to $\tilde{U}$ than to U. We therefore update the similarity of all remaining sentences with respect to $\tilde{U}$, and repeat this procedure until we fully explain U.*

Example 3 above assumed a comprehensive representation of the extraction task; however, having such representation is often prohibitive, as it would require knowing a priori all distinct aspects of the task. Our approach starts from a small, initial representation for the extraction and grows this representation gradually, as the extraction process progresses and reveals relevant aspects of the extraction task at hand. We refer to the representation of the extraction task in our approach as the *useful information* representation. Similar to earlier efforts for efficient information extraction (e.g., QXtract [AG03], PRDualRank [FC11], and FactCrawl [BLNP11a]), we can obtain an initial set of useful sentences by running the information extraction system over a small document sample. We can then produce an initial useful information representation from these useful sentences and start the sentence ranking process. Subsequently, we can process the sentences in order, until all aspects of the initial representation have been explained. Unfortunately, not all relevant aspects of the extraction task will be represented in this initial representation as it is obtained from a purposely small document sample. We can thus enhance the useful information representation by including all recently processed useful sentences, and restart the ranking process to prioritize the remaining sentences. We hope that new aspects of the task are revealed as we process more useful sentences along the extraction process.

The strategy above poses several challenges. First, the representation of sentences and useful information should be effective— to precisely characterize usefulness and novelty—as well as compact and efficient to obtain—to keep the ranking overhead to reasonable levels

[5] Removing the most correlated element is the key idea behind the Matching Pursuit (MP) algorithm [MZ93]. OMP and its grouped variant Group OMP consider all previously selected elements as well.

(Section 7.2.3). This effectiveness requirement is complicated by the fact that sentences tend to be short, so the "signals" that we can obtain from them are rather limited. The efficiency requirement is complicated by the fact that approaches for modeling sentences in a rich and compact manner (e.g., Paragraph Vector [LM14]) require substantial computation for each sentence. Second, sentence scoring and ranking should also be lightweight and efficient, to obtain novel sentences faster than by processing the sentences plainly and exhaustively (Section 7.2.4). Finally, our approach should allow trading relevance for novelty in a robust manner, as in the related areas of information retrieval and recommender systems, to support different application requirements (Section 7.2.5). The following sections describe the details of our approach.

7.2.3 Modeling Sentences and Useful Information

We now define how we model sentences and useful information to enable effective and efficient sentence ranking for information extraction.

Modeling Sentences: As described in Section 7.2.2, we need a compact sentence representation that enables efficiently and effectively characterizing sentence usefulness and novelty. One alternative would be to resort to traditional sentence representation strategies, and model a sentence as a sparse bag-of-words vector over the space of words in the documents. Although this (sparse) representation leads to efficient Group OMP solutions[6], its effectiveness is limited: A sentence will only correlate to the useful information when at least one of the words in the sentence is explicitly modeled in the useful information representation. This representation, for instance, would fail to correlate a sentence that includes the word "quake," a synonym of "earthquake" and hence very related to natural disasters, with a representation of the useful information that includes the word "earthquake."

A more effective representation that alleviates the limitations above is to exploit the so-called distributed word representations, namely, semantically rich, dense word vectors, and model a sentence as a set of dense vectors. Specifically, for a given set of training documents (e.g., Wikipedia [Wik15]) and a dimension m for the vectors, approaches for learning distributed word representations (e.g., Word2Vec [MCCD13] or Glove [PSM14])

[6]Only non-zero components will be considered during Group OMP.

produce dense vectors for all words such that words with similar meaning appear near (e.g., with respect to cosine similarity) in an m-dimensional space. For example, the vectors for "earthquake" and "quake" would be very similar, which is valuable for our *Occurs-in* task. These vectors are generally learned by performing dimensionality reduction techniques, such as matrix factorization, over some representation of the co-occurrence of words in the training documents, such as the co-occurrence matrix. For even richer representations, we can obtain a set of meaningful n-grams (e.g., "Richter scale" or "tropical storm") from an external collection following the approach in [MSC+13] and learn vectors for them.[7] (We consider these sentence representations in our experimental evaluation.) In summary, for a given sentence $s = w_1 w_2 \dots w_n$, our final representation s_M of s is $s_M = [\vec{t_1}, \vec{t_2}, \dots, \vec{t_t}]$, $t \leq n$, where t_i is an n-gram from the training collection included in the sentence and $\vec{t_i}$ is its learned vector stored as a column vector. This representation is often referred to as bag of n-grams, because the order of the n-grams in the sentence is disregarded.

Importantly, the benefits of adopting the (dense) bag of n-grams representation above are twofold. First, unlike the sparse bag-of-words approach, assessing the relevance of a sentence is possible even when its n-grams have not been explicitly modeled in the useful information. This is particularly important at early stages of the extraction process, when not much information about the extraction task has been revealed. Second, the novelty of sentences can be assessed in a robust manner, because the components of the dense vectors carry semantically rich evidence of the n-grams.

Modeling Useful Information: Beyond modeling sentences we need to find a compact and effective representation U for the useful information, as argued in Section 7.2.2. One approach is to use the n-grams in useful sentences in a similar fashion to that for the sentences above: U would consist of learned column vectors for all unique n-grams in the already processed useful sentences. However, although this representation will comprehensively cover all (seen) aspects of the useful documents, this representation has two crucial limitations. First, efficiency will be compromised, because U will grow as the extraction process progresses, and will require substantially more computation during sentence scor-

[7] This method decides whether words w_i and w_j should form a bigram by computing $\frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)}$, where $count(t)$ is the frequency of token t in the training collection and δ is a discounting coefficient to prevent large numbers of bigrams with rare words.

ing and ranking. Second, accuracy will also be compromised, because not all n-grams in the useful sentences may be equally discriminative for the task. For example, the word "tsunami" in the useful sentence *"A tsunami swept the coast of Hawaii"* for *Occurs-in* is intuitively more discriminative than words "coast" or "Hawaii." As a result, correlating to these less-discriminative n-grams may lead to prioritizing useless sentences, which is undesirable.

An alternative approach, and one that alleviates the limitations above, is to perform dimensionality reduction over the useful sentences. There are two broad families of dimensionality reduction methods that we can use in our problem. The first family consists of *feature selection* methods [MBN02], where the goal is to identify a small subset of discriminative n-grams from the available n-grams. Such methods would identify, for instance, that the word "tsunami" is more discriminative than words "coast" and "Hawaii" for the *Occurs-in* relation and should hence be modeled in the useful information. Interestingly, selecting the n-grams for the useful information—initially and after an update—can be done efficiently, provided the adopted feature selection method is efficient. In the course of our experiments we adopt a feature selection approach based on ℓ_1-norm regularized Support Vector Machines (SVM) [BBE+03]. Specifically, this approach trains an SVM linear classifier and uses the (few) non-zero learned weights to assess the relevance of the feature to the task. This approach manages to remove noisy and redundant features, thus effectively modeling the different aspects of the extraction task at hand.

The second family consists of *feature extraction* methods [Fod02], in particular, the (sub)class of *signal representation* methods. Here, the goal is to transform a high-dimensional feature space into a different lower-dimensional feature space without loss of information. In our experiments, we used Principal Component Analysis (PCA) [Shl05] to obtain such lower-dimensional space. Specifically, we perform PCA over $M \in \mathbb{R}^{m \times t}$, namely, the m-dimensional representation of all N unique n-grams in the useful sentences, to obtain an m-dimensional representation for U as follows: (i) compute covariance matrix $K \in \mathbb{R}^{m \times m}$ of M, defined as $K = \frac{1}{N} M M^T$; (ii) compute the eigenvectors of K, which by definition will have m dimensions; and (iii) use the computed eigenvectors as column vectors for U. Each eigenvector e_i is associated with an eigenvalue λ_i whose absolute value $|\lambda_i|$ indicates the

Symbol	Type	Description
$\|.\|$	$\mathbb{N}$	Size operator
$\|\|.\|\|_2$	$\mathbb{R}$	ℓ_2 norm
$\backslash$	—	Regression solve operator
E	—	Information extraction system
s	$\mathbb{N}$	Number of sentences
n	$\mathbb{N}$	Number of unique n-grams
m	$\mathbb{N}$	Dimension of dense vector
l_j	$\mathbb{N}$	Length of the j^{th} sentence
l_{max}	$\mathbb{N}$	Maximum length of a sentence
y	$\mathbb{R}^m$	*Centered* right-hand side
X	$\mathbb{R}^{m \times n}$	*Centered and normalized* n-gram matrix
X_j	$\mathbb{R}^{m \times l_j}$	*Centered and normalized* matrix for the j^{th} sentence
K	$\mathbb{N}$	Maximum number of iterations
ϵ	$\mathbb{R}$	Precision for stopping
J_{sel}	$\{\dots\}$ (set)	Selected sentences
β	$\leq \mathbb{R}^k, 0 \leq k \leq K l_{max}$	Coefficients for linear system

Table 7.1: List and description of symbols for Group OMP.

importance in the covariance matrix.

7.2.4 Scoring and Ranking Sentences via Group OMP

Based on the definition of sentences and useful information described above, we now formally describe our Group OMP-based algorithm. We first represent a sentence adopting a bag of n-grams strategy with learned dense column vectors for each n-gram in the sentence. Then, considering each sentence as a group of variables, we perform sparse group variable selection via Group OMP by regressing against the columns of our useful information representation, which we refer to as the *right-hand sides*. This regression against multiple right-hand sides, namely, the so-called *multitask* regression, consists of regressing against each column individually to in turn compute a unified cost, as we will see. Our Group OMP execution differs from traditional Group OMP in that we update the task only after selecting useful sentences. Specifically, we process each selected sentence with the information extraction system at hand and update the task only if the sentence is useful for the extraction task at hand. Upon convergence, our approach builds an updated version of the useful information that uses the processed sentences and restarts Group OMP—over the

Algorithm 2: Group OMP for Information Extraction

Input: y, X, s, E, K, ϵ.
Output: J_{sel}.

$r^0 \leftarrow y$, $J^0_{sel} \leftarrow \emptyset$, $J_{cand} \leftarrow \{1 \dots s\}$;
for $k \in 1, 2, 3, \dots, K$ **do**
  $j^k \leftarrow \texttt{argmin}(X, r^{k-1}, E, \epsilon, J_{cand})$; `/* Get next useful sentence */`
  $X^k_{aug} \leftarrow \texttt{augment}(X, J^{k-1}_{sel} \bigcup j^k)$;
  $\beta^k \leftarrow X^k_{aug} \backslash y$;
  $r^k \leftarrow y - X^k_{aug}\beta^k$; `/* Update right-hand side */`
  $J^k_{sel} \leftarrow J^{k-1}_{sel} \bigcup \{j^k\}$;
  $J_{cand} \leftarrow J_{cand} - \{j^k\}$;
  if $||r^k||_2 \leq \epsilon$ **then** `/* Evaluate convergence */`
    break
return J^k_{sel};

non-processed sentences with the updated right-hand sides. For simplicity, our description assumes that we select only the best, most correlated useful sentence during each Group OMP iteration and that we regress against a single right-hand side (i.e., a single column of our useful information representation). As we will see, the extension to multiple selected sentences per iteration as well as to multiple right-hand sides is rather straightforward. For reference, we summarize the list of symbols used in the description in Table 7.1.[8]

Sparse Group Selection for Sentence Ranking: We begin by describing the Group OMP algorithm [LSA11], which we use for sparse group variable selection (Algorithm 2). This algorithm receives as input y, X, s, E, K, and ϵ (see Table 7.1). Briefly, $y \in \mathbb{R}^m$ is one right-hand side, specifically, a column of the useful information representation; $X \in \mathbb{R}^{m \times n}$ is the matrix representation of all the sentences of interest, which consists of the vectors of all unique n-grams in the sentences;[9] s is the number of sentences in the input documents to process; E is the information extraction system of choice; and K and ϵ determine the convergence of Group OMP, respectively, by reaching a maximum number of iterations or by determining that the right-hand side has been sufficiently "explained." The objective is

[8] For purposes of numerical stability, during modeling each column (word vector) of each sentence is both mean centered and scaled by the column standard deviations. Also, the right-hand side y is only centered.

[9] As we will see Group OMP requires access to one sentence X_j at a time. We use a vector of indexes over this matrix to obtain the bag-of-n-grams representation for a sentence.

to find a sparse set of sentences that solve the system $X\beta = y$ by enforcing group penalty over the sentences.

Algorithm 2 approximates ℓ_0-sparse, group-penalized solution by iteratively choosing the useful sentence that is most correlated to the right-hand side based on the residual. The first step of the algorithm initializes the residual r^0 as the input right-hand side, the set of selected sentences, and the set of sentences to evaluate (line 1).[10] The first step in sentence selection consists of an `argmin` function (see Algorithm 3) that goes over each eligible sentence and evaluates correlation of the sentence to the current residual (line 3). This step is the performance critical portion of the algorithm, as we discuss in Section 7.2.6. Furthermore, in `argmin` the evaluation is done against the current residual r^{k-1}, and not the original right-hand side y, to promote novel sentences that have low correlation with respect to previously selected sentences while having high correlation with the useful information representation. At the end of the first step, the sentence j^k, which is the most likely to be novel, is available. The next step is to re-estimate the model coefficients by adding sentence j^k to the model (lines 4-6). Given our representation of sentences (see above), the function `augment` can be done efficiently, by simply appending the n-grams in sentence j. Then, the set of selected sentences J_{sel} and candidate sentences J_{cand} is updated (lines 7-8). Finally, if the ℓ_2 norm of the residual[11] is smaller than the precision for stopping ($||r^k||_2 \leq \epsilon$), the iterations are terminated (lines 9-10). (We later explain how we evaluate for convergence when r^k consists of multiple right-hand-sides.) By definition, the smaller the ℓ_2-norm of a vector, the closer the vector is to $\vec{0}$, the 0 vector. For our problem, $\vec{0}$ indicates that all aspects of the initial useful information matrix have been sufficiently explained. Note that Group OMP also stops when it reaches the maximum number of iterations K. (For a detailed analysis of the convergence and consistency of Group OMP, please see [LSA11].)

Selecting Novel Sentence: The implementation of the `argmin` function is given in Algorithm 3. This algorithm considers each eligible (i.e., yet unselected) sentence and evaluates its suitability to be added to J_{sel}. For each eligible sentence j, the matrix X_j corresponding

[10] We use superscript to denote the iteration number through Algorithm 2 (e.g., r^0 represents the residual at iteration 0).

[11] ℓ_2-norm of a vector x is $||x||_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$.

Algorithm 3: `argmin`

Input: X, r, E, J_{cand}.
Output: j_{sel}: Selected sentence index.

1 $C \leftarrow \emptyset$;
2 **for** $j \in J_{cand}$ **do** `/* Evaluate sentences */`
3 $\quad \beta_j \leftarrow X_j \backslash r$;
4 $\quad r_j \leftarrow r - (X_j \beta_j)$;
5 $\quad C \leftarrow C \cup \{(j, cost(r_j))\}$
6 $sortAscending(C)$; `/* Order sentences by cost */`
7 $j_{sel} \leftarrow -1$;
8 **for** $i \in 1, 2, 3, \ldots, |J_{cand}|$ **do** `/* Find best useful sentence */`
9 $\quad$ **if** $isUseful(E, j_{C[i]})$ **then**
10 $\quad\quad j_{sel} \leftarrow j_{C[i]}$;
11 $\quad\quad$ **break**;
12 **return** j_{sel}

to this sentence is used to compute the residual r_j by first regressing r for X_j (line 3), which involves solving a dense linear system, and then computing the residual (lines 4). Finally, the goodness of fit of sentence j, defined in terms of $cost(r_j)$ (i.e., cost of the residual r_j), is evaluated and stored to be compared with that of other sentences (line 5). Notice that the evaluation of sentences is highly parallelizable if $cost(r_j)$ is both associative and commutative. (Below we define the cost functions that we use in our approach, which are both associative and commutative.) The algorithm then prioritizes the sentences according to their cost (line 6). In a subsequent step, this algorithm processes the sentences in order (lines 8-11) until it finds a useful sentence. As discussed, a sentence is useful if it produces tuples when processed with the information extraction system E. The algorithm returns the best useful sentence, in terms of its goodness of fit.

Note that in the algorithm above the information extraction system may run over all sentences when there are no useful sentences in the input data. There are several possible improvements of Algorithm 3 to speed up computation: One such improvement would be to only forward in line 4 the sentences that "explain" a certain amount of the residual r. This is often done by comparing the residuals r_j and r (e.g., based on $||r - r_j||_2$, the ℓ_2-norm of their difference, $r - r_j$).

Goodness of a Sentence: As argued above, the candidate sentence to return from each

iteration is determined based on its goodness of fit. Intuitively, the best X_j is the one that explains most of r, which in turn would reduce $||r - X_j\beta_j||_2$. In fact, the best choice would correspond to an X_b that can fully explain r (i.e., $r - X_b\beta_j = \vec{0}$). Several equally effective cost functions for measuring goodness of fit have been proposed in the literature, from which we use Mean Squared Error (MSE). We describe MSE as well as other valuable functions in terms of r, X_j, and β_j next:

- *Mean Squared Error (MSE):* MSE measures the average of the squares or the errors in the estimate $\hat{r} = X_j\beta_j$ when compared to r. Intuitively, MSE indicates how much of the right-hand side r remained "unexplained" after regression. Specifically,

$$MSE(\hat{r}) = \frac{||r - \hat{r}||_2^2}{DoF}, \tag{7.1}$$

  where DoF, or degrees of freedom, is another cost measure, which we define next.

- *Degrees of Freedom (DoF):* DoF evaluates the domain of β_j. That is, DoF computes the number of components of β_j that need to be known to fully determine β_j. Specifically,

$$DoF(\beta_j) = m - p_{\beta_j}, \tag{7.2}$$

  where m is the dimension of the word vector and p_{β_j} is the number of non-zero components in β_j. In practice, when m is small $m - p_{\beta_j}$ is often replaced by $m - 1$.

- *Bayesian Information Criterion (BIC):* BIC is used for model selection in statistics to prevent overfitting. When many learned models are equally effective, the model with the lowest BIC—hence least likely to overfit—is preferred. Formally,

$$BIC(\beta_j) = \ln(MSE) + \left(|\beta_j|\frac{\ln(m)}{m}\right), \tag{7.3}$$

  where m and MSE are defined as above and $|\beta_j|$, the length of β_j, denotes the upper bound on the number of modeling parameters.[12]

[12]We can get the accurate number of free parameters in a model $X\beta = y$ by solving trace$(X\beta y^+)$, where y^+ is the Moore-Penrose pseudo-inverse of y [Moo20; Pen55; Pen56].

However, applying the functions above in our problem directly may be problematic, because there would be an undesirable bias towards long sentences. In fact, long sentences include a large number of n-grams that may not necessarily be correlated to the task at hand, but may explain several aspects of the useful information collectively. As a result, the cost of the residual for these (long) sentences will be low and will hence be prioritized. This problem is similar to that for long documents in early information retrieval systems: Long documents include more words and phrases—and with higher frequencies—than short documents; as a result, some scoring functions assign higher topical relevance scores to long documents than they do to short documents with comparable contents. It has been shown that penalizing for document length leads to better-quality retrieval systems [SBM96].

We build on the idea of *Pivoted Unique Normalization* proposed in [SBM96] to "normalize" the cost for a sentence. In particular, we use the formulation that accounts only for number of terms, because we consider unique terms in our sentence representation (see Section 7.2.3): We adjust the computed cost of a sentence s_j by multiplying by $(1-b)+b\times\frac{|s_j|}{|s|_{avg}}$, where $|s_j|$ is the length of s_j, specifically, the number of unique n-grams in s_j, $|s|_{avg}$ is the average length of all sentences, and b weighs length over relevance. (In our experiments we use $b = 0.25$, the best value reported in [SBM96], and $|s|_{avg} = 17.5$, a common value across diverse text collections.) Based on this formulation: (i) sentences that are longer than the average-length sentence will be penalized (i.e., their adjusted cost value will be higher); (ii) sentences that are shorter than the average-length sentence will be promoted (i.e., their adjusted cost will be lower); and (iii) sentences of average length will preserve their cost. Such weighing of sentences is possible, because for many extraction tasks the sentence length distribution is similar for useful and useless documents.

Extending to Multiple Right-Hand Sides: Our above definition of goodness of fit involved computing the cost of a residual $\hat{r} \in \mathbb{R}^m$, defined as $\hat{r} = X\beta$, because our formulation consisted originally of a single right-hand side $y \in \mathbb{R}^m$. To extend to multiple h right-hand sides (i.e., $Y \in \mathbb{R}^{m\times h}$), the goodness of fit needs to be computed over a residual $\hat{R} \in \mathbb{R}^{m\times h}$, defined as $\hat{R} = X\beta$. Note that in multitask regression the regression step reduces to solving h single right-hand side systems, as discussed, and the key challenge is on evaluating the goodness of fit of the residual $\hat{R}$ matrix. In our experiments, we evaluate three cost

functions, namely, *Minimum*, *Maximum*, and *Average*, to assess the goodness of fit of a sentence. As we see next, given a sentence j these functions are aggregated over the cost of the individual components $\hat{R}_j(i)$ of the residual $\hat{R}_j$ using any of the formulation above. Also, as before, the best candidate sentence is the one with the lowest value. We first define the basic form of this cost measures to in turn introduce their generalized, weighted variants.

- *Minimum (Min):* Min computes the cost of each individual component and uses the smallest value (i.e., the smallest residual component) as a surrogate of its aggregate cost. Specifically,

$$Min(\hat{R}) = \min_{i \in 1...h} cost(\hat{R}(i)). \tag{7.4}$$

  Intuitively, Min will lead to picking a sentence if it explains (at least) one component of Y sufficiently.

- *Maximum (Max):* Max computes the cost of each individual component and uses the highest computed value (i.e., the cost of the worst explained component) as a surrogate of its aggregate cost.

  Specifically,

$$Max(\hat{R}) = \max_{i \in 1...h} cost(\hat{R}(i)). \tag{7.5}$$

  Intuitively, Max will lead to picking a sentence that explains the components of Y with high coefficients (i.e., the components that are not yet well explained).

- *Average (Avg):* Avg computes the cost of each individual component and returns the average over the computed values as a surrogate of its aggregate cost. Specifically,

$$Avg(\hat{R}) = \frac{1}{h} \cdot \sum_{i=1}^{h} cost(\hat{R}(i)). \tag{7.6}$$

  Intuitively, Avg will lead to picking a sentence that will explain most components of Y.

Note that the functions above assume that all right-hand sides contribute equally to the extraction task. However, not all components of the useful information matrix will be

equally valuable for the task. For example, the n-grams in our useful information representation based on feature selection originally include learned weights that describe how discriminative they are for the task. Likewise, each component in our useful information representation based on feature extraction will be associated with a value (an eigenvalue) that indicates its importance for the extraction task via its absolute value. We exploit these values in generalized, weighted variants of the functions above. Specifically, the cost of each component is now weighted with its associated value (i.e., learned weight for the feature selection-based representation and eigenvalue for the feature extraction-based representation). Because these basic and weighted functions still suffer the length bias discussed above, we apply the described length adjustment to the final cost value.

Finally, the convergence criterion (see lines 9-10 in Algorithm 2) needs to be modified as well when the right hand sides comprises multiple columns. The extension to the single right-hand-side version discussed above consists of assessing if the average of all ℓ_2-norm values computed over each column of the residual is smaller than ϵ. In addition to this convergence criterion, we also evaluate that the current iteration contributes to explaining the useful information. We do this by computing the *gain* of the current iteration, namely, the cost difference between the current and last iteration, and evaluating that this gain is lower than a given threshold.[13]

7.2.5 Trading Relevance and Novelty

As discussed, providing for means of weighing relevance and novelty can benefit a broad range of downstream applications. In fact, such weighted schemes are crucial in areas such as information retrieval and recommender systems, as discussed. In our Group OMP implementation we can trade relevance and novelty by allowing a flexible number of sentences N to be pick per iteration.

The discussion above presented a Group OMP implementation that selects one useful sentence per iteration. To return N sentences per Group OMP iteration, we need to redefine our `argmin` function. We achieve this by processing sentences with the information extraction system (lines 8-11 in Algorithm 3) until we process N useful sentences. The Group

[13] In our experiments, we stop when the current gain is smaller than 0.0000001.

OMP algorithm (Algorithm 2) will now augment X^k_{aug} using the N selected sentences, and the rest of the algorithm will remain unchanged.

In practice, different values of N will have distinct implications in the relevance, novelty, and efficiency of the extraction output. Small N values will promote novelty, because updates of the useful information—hence prioritization of novel sentences—will happen frequently along the extraction process. These frequent updates, however, may impact the efficiency of the process, because a series of regressions over all sentences will have to be performed against the updated right-hand side. Large N values, on the contrary, will promote relevance because sentences will be mainly correlated to outdated version of the useful information, as updates will happen rarely. The rather rare useful information updates will lead to fewer regressions along the extraction process, favoring efficiency: Simply put, the number of regressions will be reduced by a factor of N in comparison to our earlier described ($N = 1$) algorithm. In this case, the cost of the extraction process will be dominated by the cost of running the information extraction system over a sentence. We next provide a more detailed analysis of our approach.

7.2.6 Efficiency of Our Approach

As argued above, the critical part of Algorithm 2 is the `argmin` function, which repeatedly solves a least squares problem—$\beta_j = X_j \backslash r$—and in turn computes the residual norm of this solution (see MSE above). From among the traditional matrix factorization methods to solve least square problems, namely, Cholesky, QR, and SVD [Str93], Cholesky is the most efficient. We solve the least squares problem using Cholesky factorization.

Briefly, the goal of a least squares problem is to minimize $||r_j||_2^2 = ||X_j\beta_j - r||_2^2$, where β_j is the solution to $X_j \backslash r$. As $||r_j||_2^2 = r_j^T r_j$, this problem can be rewritten as minimizing $(X_{G_i}\beta_j - r)^T(X_{G_i}\beta_j - r)$. By taking the first derivative with respect to β_j and setting it to zero, we get $X_j^T X_j \beta_j = X_j^T r$. This equation can be now solved using Cholesky decomposition, provided X_j has full rank.[14] The reason Cholesky decomposition is appealing is because the dimensions $\mathbb{R}^{l_j \times l_j}$ of the matrix $X_j^T X_j$, where l_j is the length of the j^{th}

[14]We ensure the full rank of X_j by adding a small ridge λ along the diagonal of $X_j^T X_j$.

sentence, are small.[15] We can therefore precompute and store the Cholesky factors $X_j^T X_j$ for all the sentences.[16] The backsolve is (relative to the factorization) inexpensive, as it involves $O\left(l_{max}^2\right)$ operations; therefore, the time consuming portion is computing $X_j^T r$, which involves $O\left(m l_{max}\right)$ operations. By keeping m small, by using low-dimension word vectors, we can also keep the cost of $X_j^T r$ low.[17] Furthermore, these operations are performed over h right-hand sides: by keeping h small (e.g., using the feature selection and feature extraction methods described earlier in this section), we can keep the overall cost low.

Finally, computing $X_j^T r$ is bandwidth bound, which precludes obtaining both peak throughput on an individual core and linear speedup on multi-cores. This is because computing each element of $X_j^T r$ requires m scalar multiplies and $m-1$ scalar additions, for a total of $2m-1$ operations per element. The total count of floating operations would be bounded by $l_{max}(2m-1)$. From a memory use perspective, computing $X_j^T r$ requires reading both X_j^T and r once and writing out at most l_{max} scalar values: The total number of memory operations is $l_{max}m + m + l_{max} \approx l_{max}(m+1)$. In short, the ratio of memory operations to compute operations is ≈ 1, which means that the limiting factor in throughput and speedup would be the speed with which operands can be loaded and stored.

7.3 Experimental Settings

We now describe the experimental settings for the evaluation of our adaptive ranking approach:

Collections: We used 360,000 random documents from the *NYT Annotated Corpus* [San08], which originally contains 1.8 million New York Times articles from 1987 to 2007. (We used the entire NYT corpus for our evaluation in Chapter 6 for the document ranking problem.) We split this set of documents into a tuning set (97,258 documents, for a total of 2,826,631 sentences) and a test set (262,742 documents, for a total of 7,613,296 sentences). We evaluated different combinations of techniques and parameters on the tuning set. Additionally,

[15] The average sentence length that we observed across many different text collections was 17.5.

[16] This factorization step is still considerably more efficient that computing the Paragraph Vectors of the sentence [LM14].

[17] In our experiments we observed that $m = 50$ produces high-quality word vectors.

as in Chapters 4 through 6, we used collections 1-5 from the TREC conference [TRE00] (totalling 1,038,957 unique documents) to generate the queries for the query-based sample generation that we explain later in this section. Finally, we used the Wikipedia in English dump dated 08-04-2015 (1.6 billion tokens) for training the distributed word representation algorithms.

Entity and Relation Extraction Systems: To include a variety of extraction approaches, we considered different relation extraction systems for each relation (see next), as well as different entity extraction systems for their corresponding entities, as follows:

- *Relation Extraction:* To extract our relations, we trained four relation extraction systems using REEL (see Chapter 3) and used OpenCalais [Ope15a], an off-the-shelf online service for information extraction. Specifically, the four trained relation extraction systems that we use in our experiments are: Subsequence Kernel [BM05b] (SSK), Shortest-Path Kernel [BM05a] (SPK), Bag of n-grams Kernel [GLR06] (BONG), and Dependency Graph Kernel [TPL10] (DG). We used two of these trained extraction systems, namely, SSK and BONG, in previous chapters as well.

- *Entity Extraction:* To extract the entities in our trained relation extraction system, we use the entity extraction systems from previous chapters, which we obtained as follows: We selected the best performing combination of entity extraction systems for each entity type via 5-fold cross validation over a set of manually annotated documents, and used it across all extraction tasks. However, for diversity, whenever we had ties in performance, we selected the (arguably) less common contender. Specifically, to extract *person* and *location* entities, we used the StanfordNLP named entity tagger [Sta15b]; for other entities, we trained our own entity extractors using E-txt2DB [Etx12]. Our final models are Maximum Entropy Markov Models [MFP00] for *natural disasters* and Conditional Random Fields [ML03] for the remaining entities. The entities for the relations extracted using OpenCalais are automatically extracted by the service. Importantly, OpenCalais normalizes entities from many of its entity types (e.g., person or organization), which provides for a robust evaluation of novelty between tuples, as we will see.

Relations: We include an extensive and diverse set of relations in the experiments. We use our trained relation extraction systems to extract 5 relations, namely, Person–Career, Natural Disaster–Location, Man Made Disaster–Location, Person–Charge, and Election–Winner. We also use OpenCalais to extract 83 relations that cover financial, business, sports, politics, and other relevant domains[18], for a total of 85 unique relations. The percentage of useful sentences for all relations over our tuning set ranges from 0.0004% to 27.76% (0.68% on average).

Sampling Strategies: We compared two techniques to collect the initial document sample for our initial useful information representation:

- *Random:* Random picks 2,000 documents at random from the collection.
- *Cyclic*: Cyclic (see Section 4.2.3) iterates repeatedly over a list of queries and collects the unseen documents from the next K documents that each query retrieves until it collects 2,000 documents . We learned 5 lists of queries using sets of 10,000 random documents (5,000 useful and 5,000 useless) from the TREC collection by applying the χ^2-based method in Section 4.3, which weighs keywords in the training collection according to their Pearson's χ^2 test. We used $K = 50$ in our experiments.

n-Gram Representation: We consider different configurations for n-gram representation: (i) *n-grams:* We used unigrams, bigrams, and trigrams that appear in 5 documents or more in Wikipedia, obtained with Word2Phrase [MSC+13] (for a total of 2,380,073 unigrams, 4,540,242 bigrams, and 7,430,063 trigrams); (ii) *Vector dimension:* We varied the length m of vectors ($m \in [50, 100, 300, 1000]$); and (iii) *Learning algorithm:* We used continuous skip-gram and continuous bag-of-words models [MCCD13] and GloVe [PSM14]. We do not report results for bag-of-words model, as it was consistently outperformed by skip-gram. After learning the vectors, we remove English stopwords reported in MySQL as well as rare words (i.e., words that appear in less than 0.003% of the NYT documents) and frequent words (i.e., words that appear in more than 90% of the NYT documents).

Sentence Representation: For sentence representation we use the bag of n-grams approach described in Section 7.2.3. Specifically, for a given sentence with l unique n-grams,

[18]The full list of relations extracted by OpenCalais is listed at `http://goo.gl/nD5m8V`.

and m-sized distributed word model M (see above), we build an $l \times m$ matrix with the l vectors for the n-grams obtained from M. We only include n-grams that are learned in M (i.e., n-grams from Wikipedia that are not regarded as stopwords, rare, or frequent words).

Useful Information Representation: We consider the useful information representation methods discussed in Section 7.2.3. As discussed, we build the first target using the sentences in the initial document sample and, later, from the useful sentences that are processed as the extraction progresses.

- *All:* Uses as column vectors the learned vectors of all n-grams (see sentence representation above) in the useful sentences.
- *K-Sel:* Uses as column vectors the learned vectors of the K most discriminative n-grams, obtained by performing the ℓ_1-norm regularized SVM approach in [BBE+03] over the observed useful and useless sentences, as we explained in Section 7.2.3. We evaluated different values for K ($K \in [50, 100, 300, 500, 1000]$).
- *Summ:* Uses as column vectors the eigenvectors derived from performing Principal Components Analysis [Shl05] over the *All* representation above, as described in Section 7.2.3.

Group OMP Settings: We use the following settings for our Group OMP approach: *(1) Goodness of fit:* We use the weighted version of the aggregated cost functions discussed in Section 7.2.4, namely, *Min*, *Max*, and *Avg*. After computing the cost of the residual for a sentence, we adjust this value using the length penalization strategy based on unique length normalization, with $b = 0.25$ and $|s|_{avg} = 17.5$. *(2) Convergence methods:* For each Group OMP execution (i.e., for each updated useful information representation), we perform a maximum of 100 iterations (i.e., $K = 100$ in Algorithm 2). We also stop when the ℓ_2-norm of the residual is smaller than 0.0001 (i.e., $\epsilon = 0.0001$ in Algorithm 2).

Sentence Ranking Baselines: We compare the Group OMP-based approach from this chapter to two baseline techniques:

- *SVM:* SVM is an SVM-based state-of-the-art sentence filtering technique [BHL11] that learns an SVM-based linear classifier to decide if a sentence should be forwarded

to the information extraction system. The classifier uses words and phrases in the sentences as features. The original algorithm in [BHL11] forwards the entire document to the information extraction system. Instead, we forward the sentences according to the confidence score from the classifier, which yields efficient executions and keeps sentence recall unaffected.

- *RSVM-IE:* RSVM-IE is a variant for ranking sentences of the RankSVM-based document ranking technique described in Section 6.2.1. To train the SVM classifier in RSVM-IE, we used the settings in Section 6.3, namely, online learning based on Pegasos gradient steps [SSSS07] with elastic-net regularization [ZH05] using $\lambda_{All} = 0.1$ and $\lambda_{L2} = 0.99$.

Additionally, because RSVM-IE is driven only by usefulness, we evaluate a variant of this method that incorporates novelty via the Maximum Marginal Relevance [CG98] (MMR) method. For a given query Q, MMR selects the next document D_i according to:

$$\arg\max_{D_i \in R \setminus S} \left[\alpha \cdot Sim_1(D_i, Q) - (1 - \lambda) \cdot \max_{D_j \in S} Sim_2(D_i, D_j) \right],$$

where R is the set of documents, S is the set of already selected documents, $Sim_1(D_i, Q)$ defines the relevance of document D_i to query Q, $Sim_2(D_i, D_j)$ defines the similarity between documents D_i and D_j, and λ weighs relevance over novelty. For our problem, we regard the extraction task E as the query Q and a document D_i as a sentence S_i, and define: $Sim_1(S_i, E)$ as the score given by RSVM-IE for sentence S_i when trained for E, and $Sim_2(S_i, S_j)$ as cosine similarity between S_i and S_j, where S_j is always useful. Based on this definition, we only need to update sentence scores after processing a useful sentence.

Executions: We executed each experiment five times with different samples (i.e., five different random samples and five different sets of initial sample queries), to account for the effect of randomness in the results, and report the average of these executions.

Evaluation Metrics: We use the following metrics:

- *Useful Sentence Recall:* Useful sentence recall measures the fraction of useful sentences in the documents that have been processed at different points during the extraction

(e.g., after processing $x\%$ of the sentences) and averaged over all executions of the same configuration.

- *Unique Extraction Output Recall:* Unique extraction output recall measures the fraction of unique tuples that have been extracted at different points during the extraction process (e.g., after processing $x\%$ of the sentences) and averaged over all executions of the same configuration. We also report the recall for each individual attribute. Normalized attributes, which we indicate as (N), are compared with respect to their normalized value, whereas non-normalized attributes are compared using case-insensitive string matching.

- *CPU Time:* CPU time measures the time consumed for processing and ranking the sentences.

7.4 Experimental Results

We now present the results of the experimental evaluation of our sentence ranking approach. Specifically, we first consider the configuration choices for each building block (i.e., sample generation, target generation, sentence representation, goodness function, iteration size) separately and over our tuning set. For this, and unless otherwise indicated, our GOMP-IE technique across all tuning experiments uses the following settings: (i) Cyclic for sampling generation, (ii) unigrams (i.e., $n = 1$) for bag-of-n-grams representation, (iii) 50-dimensional Word2Vec word vectors (i.e., $m = 50$), (iv) 100-Sel (i.e., K-Sel with $K = 100$) for useful information representation, (v) weighted Avg cost function for goodness of fit assessment, and (vi) updates (of the useful information representation) every 25 useful sentences (i.e., $N = 25$). Later, for the final evaluation of our approach over the test set and against the state-of-the-art ranking strategies, we use the best configuration according to the tuning set experiments.

7.4.1 Impact of Scoring Approach

To understand the impact of using our regression-based scoring approach, we first evaluate our technique of Section 7.2.4 without updating the useful information (i.e., we use the initial

useful information along the entire extraction process) against the two baseline RSVM-IE variants (i.e., without MMR) without adaptation. From our baselines, RSVM-IE (Base-D) ranks documents and processes all sentences for each document, whereas RSVM-IE (Base-S) ranks sentences instead. Figure 7.1 shows the average recall for the Person–Career relation using the BONG extraction system. Person–Career is a rather dense relation, with 4.01% useful sentences in the tuning set. For reference, we also show the performance of a random ordering of the sentences (see black dashed line in Figure 7.1), as well as of a perfect ordering where all useful sentences are ahead of the useless ones (see red dotted line in Figure 7.1). We also show in Figure 7.2 the results for the Election–Winner relation using the SSK extraction system. Unlike Person–Career, Election–Winner is a sparse relation, with only 0.01% useful sentences in the tuning set. We observed similar results for other relations. As shown, RSVM-IE (Base-S) significantly outperforms other approaches. RSVM-IE (Base-S) effectively learns words and phrases that occur in useful sentences and, in effect, identifies useful sentences with high recall and precision: 75% of the useful sentences for Person–Career are processed within the first 15% of the sentences (Figure 7.1), whereas 90% of the useful sentences for Election–Winner are processed within 5% of the sentences (Figure 7.2). We also observe that our GOMP-IE approach exhibits similar performance to that of RSVM-IE (Base-D): This happens because the distributed word representation that we use to model n-grams leverages knowledge beyond sentence level. In turn, many useless sentences that include words correlated to those in the useful information representation are prioritized. Unfortunately, this drastically impacts the performance of GOMP-IE for sparse relations during early stages of the extraction process (see Figure 7.2 for Election–Winner), because a vast majority of the sentences are useless. We study this effect in more detail in Section 7.4.3.

7.4.2 Impact of Sampling Strategy

To understand the impact of different sampling techniques, which we used to build the initial useful information representation, we compared the Random and Cyclic sampling techniques (Section 7.3). Figure 7.3 shows the average recall for the Political Entity–Allied or Rival relation using the OC extraction system. We observed similar results across relations (e.g., Figure 7.4 shows the results for Natural Disaster–Location using the SSK

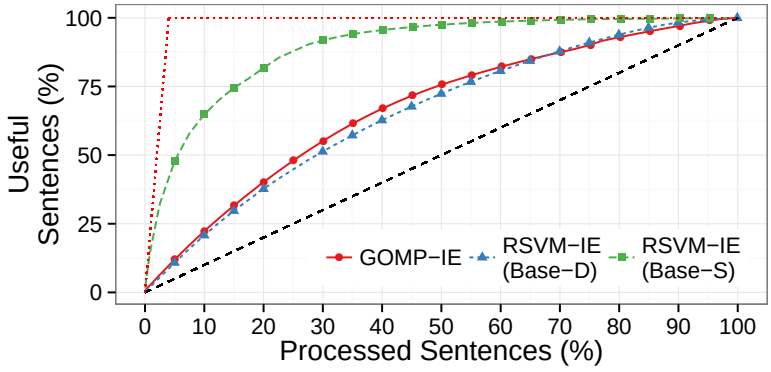


Figure 7.1: Useful sentences recall for Person–Career for different ranking generation techniques and using the BONG extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

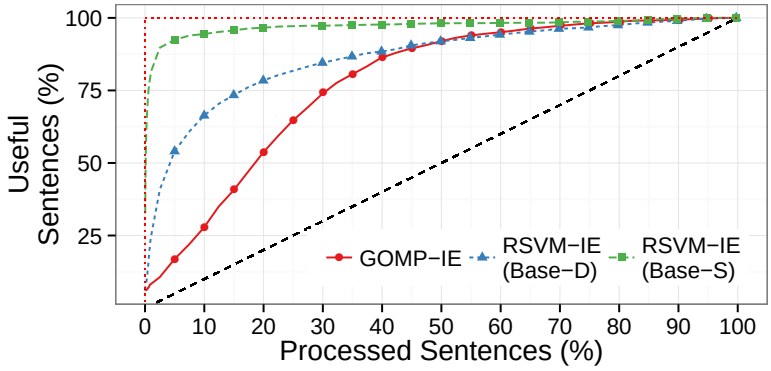


Figure 7.2: Useful sentences recall for Election–Winner for different ranking generation techniques and using the SSK extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

extraction system). As shown, the extraction process performs comparably for both sampling approaches; however, the recall of the extraction process improves more rapidly at early stages of the extraction process with the Random sampling approach than does with the Cyclic approach. This occurs because a random sample is expected to better represent distinct aspects of the extraction task, provided the (small) document sample includes useful documents. The Cyclic approach, on the contrary, may incur a certain bias towards the useful contents in the training collection. We observe this in Figure 7.4 for Natural Disaster–Location, where the execution using Cyclic starts with a higher number of use-

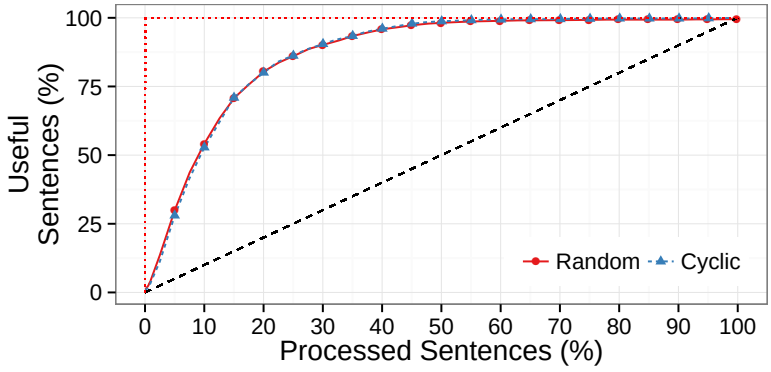


Figure 7.3: Useful sentences recall for Political Entity–Allied or Rival for different sample generation techniques and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

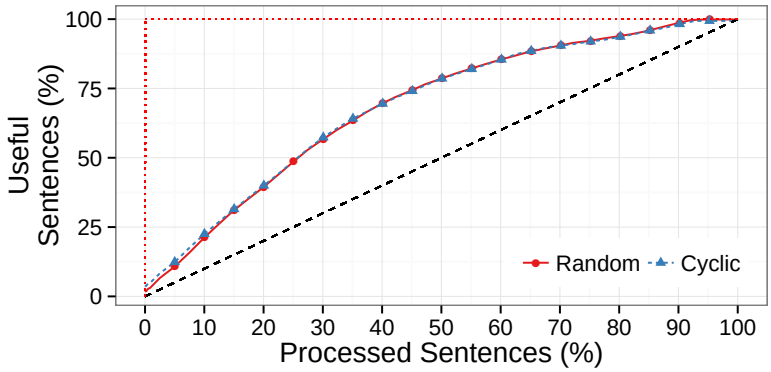


Figure 7.4: Useful sentences recall for Natural Disaster–Location for different sample generation techniques and using the SSK extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

ful documents than does Random, but these useful sentences produce the same extraction output (see below).

Beyond useful sentences recall, we also evaluate the impact of different sampling techniques on the number of unique tuples and attributes extracted along the process. Figure 7.5 shows the average fraction of unique tuples and attributes for the Political Entity–Allied or Rival relation using the OC extraction system. Other relations exhibited similar results (e.g., Figure 7.6 shows the results for Natural Disaster–Location using the SSK extraction system). As above, both sampling approaches produce comparable extraction processes.

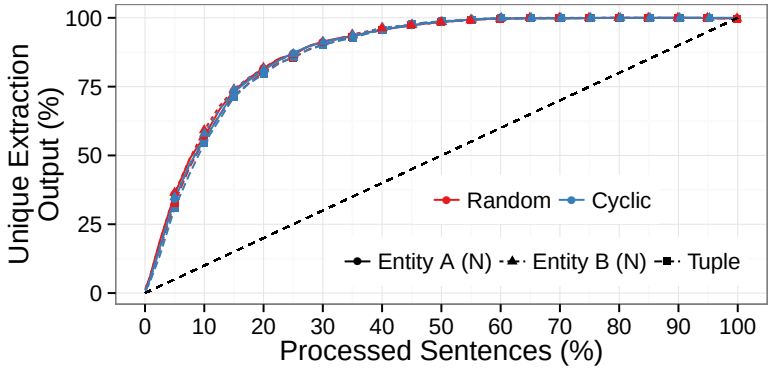


Figure 7.5: Unique extraction output recall for Political Entity–Allied or Rival for different sample generation techniques and using the OC extraction system.

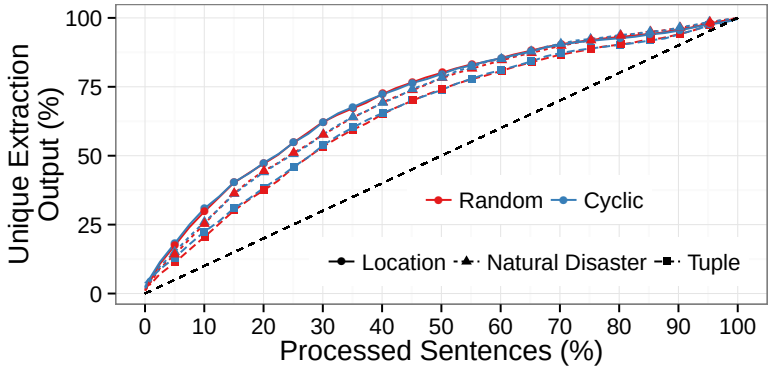


Figure 7.6: Unique extraction output recall for Natural Disaster–Location for different sample generation techniques and using the SSK extraction system.

For Political Entity–Allied or Rival, the Random sampling approach consistently exhibits better extraction output recall than the Cyclic approach. We observe a similar trend for Natural Disaster–Location even when the extraction process has collected more useful useful sentences with the Cyclic sampling approach: This is particularly evident at early stages of the extraction process (10% or fewer processed sentences) for the attribute Location. In sum, both sampling approaches yield extraction processes of similar performance. However, the Random sampling strategy should only be used when a random document sample is likely to include useful documents for the extraction task at hand.

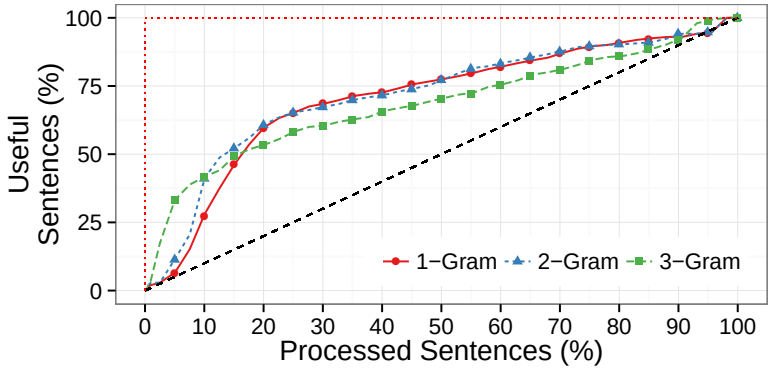


Figure 7.7: Useful sentences recall for Movie–Release Date for different lengths n of n-gram and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

7.4.3 Impact of Sentence Representation

As discussed in Section 7.3, there are three broad dimensions related to the distributed word representation that characterize the modeling of the sentence, namely, length n of n-grams, dimensions m of learned distributed vector, and learning strategy. To evaluate the impact of sentence representation, we consider the Word2Vec and GloVe learning models described in Section 7.3 and vary n and m in two independent evaluations.

Figure 7.7 shows the average recall for the Movie–Release Date relation using the OC extraction system using Word2Vec and varying n. Other relations yielded analogous conclusions. As shown, higher values for n perform better at early stages of the extraction process. This is because longer n-grams are unlikely to be ambiguous and, in effect, their vectors carry a precise, meaningful representation. However, the main limitation of high values for n is that the number of unique n-grams becomes prohibitively large, which complicates finding a small set of n-grams that comprehensively represent the extraction task. Small values for n, on the contrary, yield meaningful representations of the n-grams while keeping the number of unique n-grams relatively small. This translates into extraction processes that are consistently better than those of longer n values. Moreover, although many naturally ambiguous n-grams (e.g., "Java," "sweep," "New York," "Rio") may carry multiple meanings in their vectors, this ambiguity may have little or no impact when sentences include non-ambiguous terms as well.

In addition to the evaluation of n above, we also evaluated the impact of m, namely, the size of the learned distributed vectors. Figure 7.8 shows the average recall for the Movie–Release Date relation using the OC extraction system and varying the learning algorithm and m. We observed comparable results for other relations. As shown, the learning algorithms that we compare, specifically, Word2Vec and Glove (W2V and GV in Figure 7.8, respectively) exhibit similar performance for the same dimension size m. We observe, however, that W2V consistently outperforms GV. This corroborates results from earlier comparisons of these methods for other text-centric tasks (e.g., [LGD15]). An important observation concerns the (poor) performance of high-dimensional vectors (see GV-300 and W2V-300 in Figure 7.8), which is affected for two main reasons. First, these vectors capture characteristics of the n-grams that are beyond the rather shallow features in our information extraction systems. This produce high correlation between n-grams—from the sentence and the extraction task representation—that do not lead to the extraction of tuples. Second, these vectors lead to a large representation of the extraction task, which will only be fully explained after a large number of iterations. This drastically hurts the performance of the extraction process when the initial document sample includes only a (very) small number of useful documents, as is the case for the Movie–Release Date relation. This substantially hurts the precision of our approach. For (relatively) small vectors (i.e., $m = 50$ and $m = 100$), the performance of our approach is similar along the extraction process. Deciding the value for m therefore becomes an efficiency-related decision, which is, of course, better for small values for m. We evaluate efficiency later in this section.

We finally compare the settings above in terms of unique extraction output. Figure 7.9 shows the fraction of unique tuples and attributes for the Movie–Release Date relation using the OC extraction system using Word2Vec and varying n. Other relations yielded analogous conclusions. As shown, unique extraction output correlates with the recall from our analysis above. Specifically, using long n-grams tends to collect a higher fraction of unique tuples and attributes during early stages of the extraction process, whereas using short n-grams tends to perform best during the remaining portion of the process. This implies that the length of n-grams has little or no impact on the tuples and attributes that we collect along the extraction process. It is important to note, however, that picking n

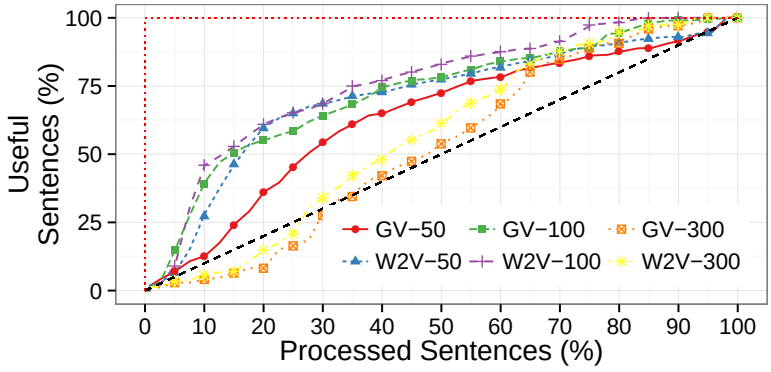


Figure 7.8: Useful sentences recall for Movie–Release Date for different dimensions m of distributed vector and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

may be an extraction task-specific choice in certain cases. Using bigrams or trigrams may produce highly meaningful sentence representations for extraction tasks that, for example, involve entities of type *person*.

Additionally, we evaluated the impact of m in unique extraction output recall. Figure 7.10 shows the fraction of unique tuples and attributes for the Movie–Release Date relation using the OC extraction system and varying m. (For clarity, we only show the best performing techniques from our analysis above, namely, GV-50, GV-100, W2V-50, and W2V-100.) As shown, modeling sentences with high-dimensional vectors has limited impact on the overall performance of the extraction process. In fact, W2V-50 and W2V-100 (i.e., Word2Vec with $m = 50$ and $m = 100$, respectively) produce comparable extraction output after processing 20% of the sentences. Moreover, and as shown above, higher values for m (e.g., $m = 300$) substantially hurt the performance of the overall extraction process. Therefore, m should be kept small for effectiveness but also, as discussed in Section 7.2.6, for efficiency.

7.4.4 Impact of Useful Information Representation

To evaluate the impact of useful information representation, we compared the variants discussed in Section 7.2.3, namely, All, K-Sel, and Summ, using the settings of Section 7.3. Figure 7.11 shows the average recall for the Company–Relation Type relation using the

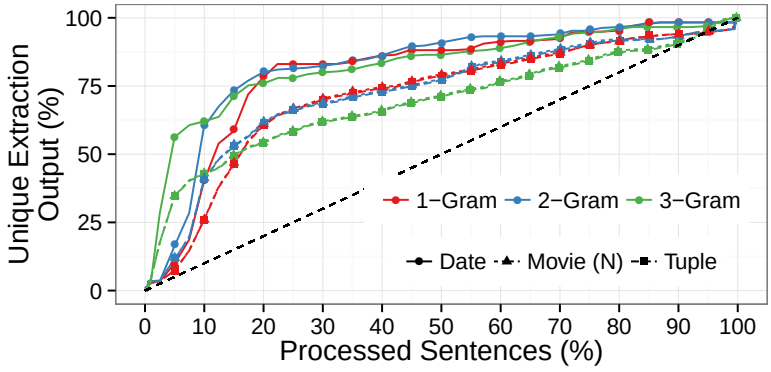


Figure 7.9: Unique extraction output recall for Movie–Release Date for different lengths n of n-gram and using the OC extraction system.

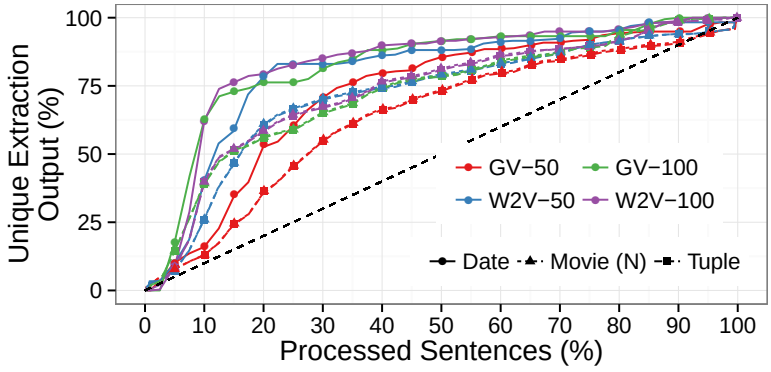


Figure 7.10: Unique extraction output recall for Movie–Release Date for different dimensions m of distributed vector and using the OC extraction system.

OC extraction system for All, K-Sel (with $K = 100$ and $K = 300$), and Summ useful information representation methods. Other relations yielded analogous conclusions (e.g., Figure 7.12 shows the results for Person–Charge using the BONG extraction system). As shown, the performance of the extraction process is highly correlated with the specificity of the useful information representation: 100-Sel, which forms the useful information with the 100 most relevant words for the extraction task, consistently outperforms all other approaches. This corroborates our conclusion in Section 7.4.1 on the (high) effectiveness of high-precision approaches. The Summ approach, which builds the useful information representation via a linear combination of different words exhibits the lowest performance. This occurs because Summ may eliminate certain dimensions that are highly relevant to the

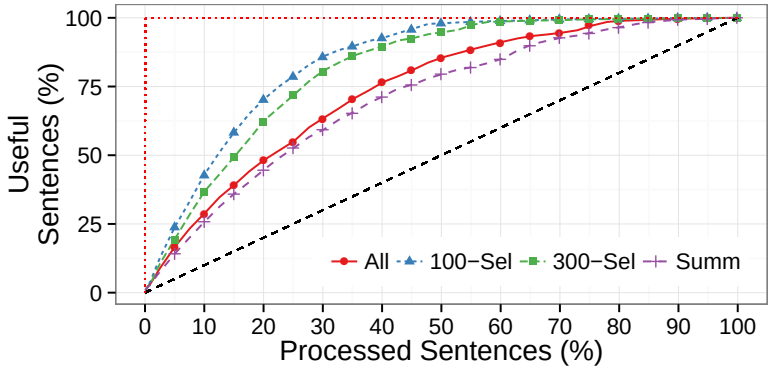


Figure 7.11: Useful sentences recall for Company–Relation Type for different useful information representation strategies and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

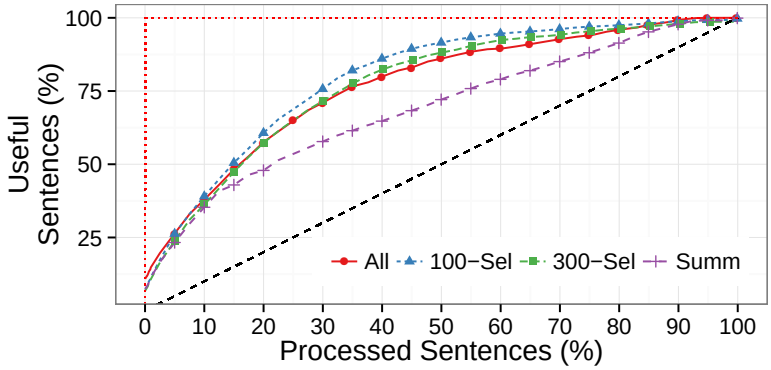


Figure 7.12: Useful sentences recall for Person–Charge for different useful information representation strategies and using the BONG extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

extraction task, as they may exhibit low variance across useful sentences, and keep other non-relevant terms instead.

We also study the impact of different useful information representations on the unique tuples and attributes that are extracted along the extraction process. Figure 7.13 shows the average fraction of unique tuples and attributes for the Company–Relation Type relation using the OC extraction system. We observed similar results over other relations (e.g., Figure 7.14 shows the results for Person–Charge using the BONG extraction system). Similarly to what we observed above for useful sentence recall, the most specific useful information

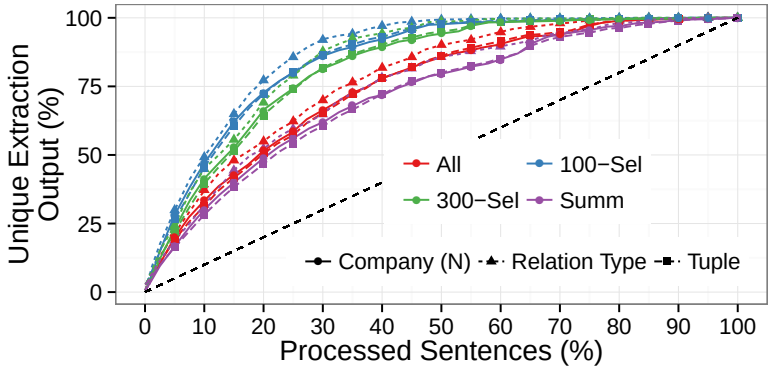


Figure 7.13: Unique extraction output recall for Company–Relation Type for different useful information representation strategies and using the OC extraction system.

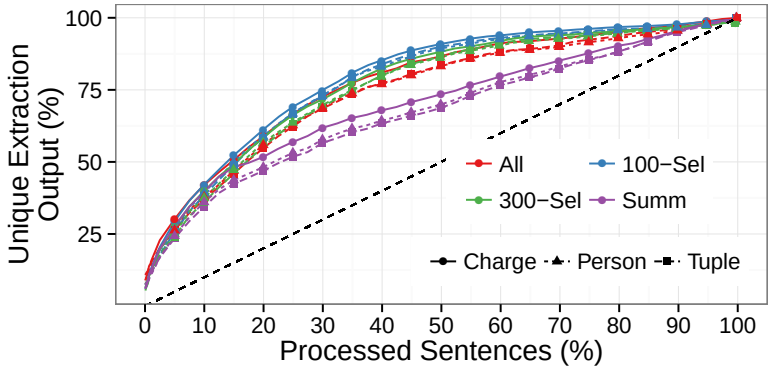


Figure 7.14: Unique extraction output recall for Person–Charge for different useful information representation strategies and using the BONG extraction system.

representations perform best. Furthermore, the difference in unique extraction output recall across techniques is consistent along the extraction process, even when the representation of strategies such as All increases as the extraction process progresses. For 100-Sel and 300-Sel, in particular, this suggests that the selected n-grams effectively represent different aspects of the extraction task at hand. As another important observation, for each technique the recall for tuples and individual attributes is comparable along the extraction process and not dominated by one single attribute. This implies that the extraction output is diverse and not predominantly about a small set of attribute values. We study this effect in more detail in our comparison with the state-of-the-art approaches (Section 7.4.8).

7.4.5 Impact of Goodness of Fit Computation

To evaluate the impact of the goodness computation, we compare the generalized, weighted variants of the cost functions described in Section 7.2.4, namely, Min, Max, and Avg. Figure 7.15 shows the average recall for the Music Album–Release Date relation using the OC extraction system. Other relations yielded analogous conclusions. As shown, Avg consistently outperforms Max and Min along the extraction process. This result corroborates our hypotheses from Section 7.2.4 for the different cost functions: Avg prioritizes sentences that sufficiently explain multiple important aspects of the extraction task, which leads to prioritizing most useful sentences early in the sampling process. For Music Album–Release Date, 90% of the useful sentences are processed within the first 20% of the sentences. Likewise, Max prioritizes sentences that explain the important aspects of the extraction task that remain largely explained in the useful information representation. This leads to prioritizing sentences similarly to Avg during early stages of the extraction process; however, when most relevant aspects of the extraction task have been explained, Max fails to continue identifying useful sentences. Also, Max exhibits a higher bias towards long sentences than those of other cost functions, since long sentences will explain most aspects of the extraction task to some extent, thus producing "flat" residuals (i.e., residual with low, comparable values). Finally, Min prioritizes sentences that explain at least one important aspect of the extraction task sufficiently. When multiple sentences fully explain at least one right-hand side (i.e., the cost of the residual for the right-hand side is 0), Min exhibits some serious limitations: Min is unable to "break ties" and the sentence ranking becomes nearly random. Min becomes more effective later in the extraction process, specifically, when useless sentences that fully explain at least one aspect of the extraction task have been already processed. The performance of Min is conditioned by how discriminative the n-grams of the extraction task at hand are.

7.4.6 Impact of Sentences per Iteration

In Section 7.2, we argued that the number of sentences N per iteration in our Group OMP-based approach, or the number of useful sentences we process before updating the useful information representation, weighs relevance and novelty. We now evaluate the impact of

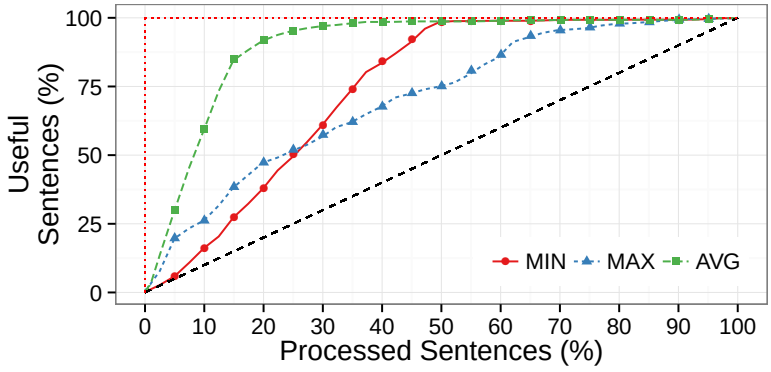


Figure 7.15: Useful sentences recall for Music Album–Release Date for different goodness functions and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

N. Figure 7.16 shows the average recall for the Endorsee–Endorser relation using the OC extraction system. Other relations yielded analogous conclusions. As shown, at early stages of the extraction process, larger values for N lead to higher recall values than those of small values for N. This occurs because large N values delay updating the useful information representation. This delay, unfortunately, has a negative impact on the overall performance of the extraction process, because the useful information representation is not frequently enhanced. Small values for N, on the contrary, lead to frequent updates of the useful information representation, for novelty (see below). In turn, this leads to enhancing the useful information representation more frequently than with high values for N. This explains why the recall values for small values for N are comparable to those of high values for N.

We now evaluate the unique tuples and attributes for different values of N. Figure 7.17 shows the average fraction of unique tuples for the Endorsee–Endorser relation using the OC extraction system. We observed similar results for other relations. As shown, all values for N exhibit comparable unique extraction output. Interestingly, this occurs even when the executions for high values of N have processed a larger number of useful sentences (see Figure 7.16), which suggests that using small values for N effectively promote novelty along the extraction process. This result corroborates the hypothesis in Section 7.2.5, namely, that our approach provides for trading relevance and novelty in a robust manner.

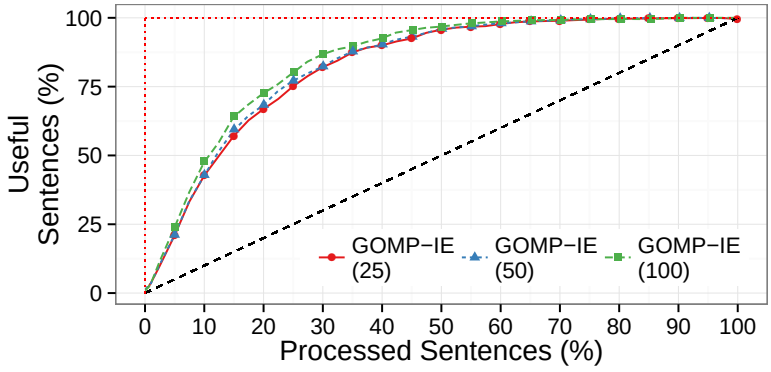


Figure 7.16: Useful sentences recall for Endorsee–Endorser for different number of sentences N per iteration and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

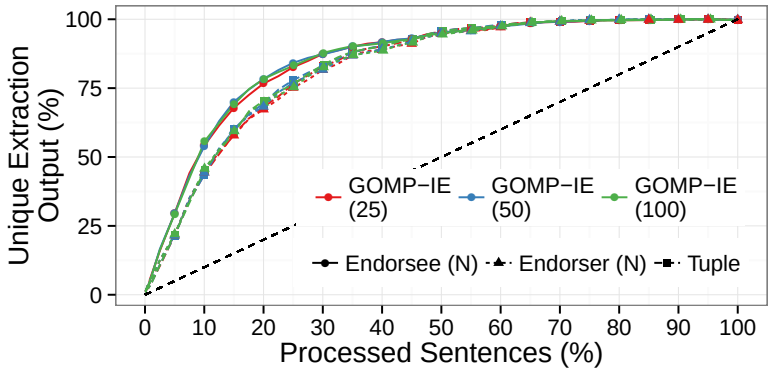


Figure 7.17: Unique extraction output recall for Endorsee–Endorser for different number of sentences N per iteration and using the OC extraction system.

7.4.7 Impact of Document Set Characteristics

The goal of this experiment is to evaluate the performance of our technique as a building block in a system using the techniques of Chapter 6, which focus on a (relatively small) set of potentially useful documents. We evaluate our best performing technique, which, according to our experiments above, consists of: (i) Cyclic for sampling generation, (ii) 100-Sel (i.e., K-Sel with $K = 100$) for useful information representation, (iii) unigrams (i.e., $n = 1$) for bag-of-n-grams representation, (iv) 50-dimensional (i.e., $m = 50$) Word2Vec word vectors, (v) weighted Avg cost function for goodness of fit assessment, and (vi) updates (of the useful information representation) every 25 useful sentences (i.e., $N = 25$). We run the

technique above over splits of documents—obtained from the tuning set—with fractions of useful documents ranging from 10% to 90%. These splits represent sets of documents to process that are retrieved by queries of different quality for the extraction task at hand.

We form a document split with a fraction p of useful documents as follows: (1) we learn 50 queries from the document sample obtained with Cyclic by following the SVM approach for K-Sel described in Section 7.3; (2) we retrieve from our tuning collection C at most 1,000 documents per query, and combine them in a single set of documents D ; (3) we update D, so that it includes $|D_+| = p \cdot |D|$ useful documents and $|D_-| = (1 - p) \cdot |D|$ useless documents. The steps from Item (1) and Item (2) are shared across different values of p. We perform the step in Item (3) for each value of p; this step consists of removing useless documents from D at random or picking documents from C at random until $|D_+| = p \cdot |D|$.

Figure 7.18 shows the average recall for the Company–Customer relation using the OC extraction system. As shown, our approach effectively prioritizes useful sentences for different fractions of useful documents. In particular, our approach performs best when the fraction of useful documents is small. This document distribution is typical when the learned queries aim to retrieve useful documents with high recall. Moreover, our approach exhibits substantial improvements over an approach that processes all retrieved documents (see black dashed line in Figure 7.18), even when a vast majority of the documents is useful. This document distribution corresponds to a set of queries that retrieve useful documents with high precision. In sum, our approach improves the efficiency of the extraction process under different precision and recall requirements, which demonstrates the merits of our approach as a building block for efficient information extraction.

7.4.8 Comparison with Baseline Ranking Strategies

We now compare over the test set our best performing configuration (see Section 7.4.7) against the SVM baseline approach in Section 7.3 and two variants of RSVM-IE: (i) RSVM-IE (Adap-S), which performs RSVM-IE at sentence level with adaptation; and (ii) RSVM (MMR-S), which incorporates MMR to form the (strong) baseline technique discussed in Section 7.3. We evaluate the techniques on useful sentence recall, unique extraction output recall, and efficiency (in terms of CPU time).

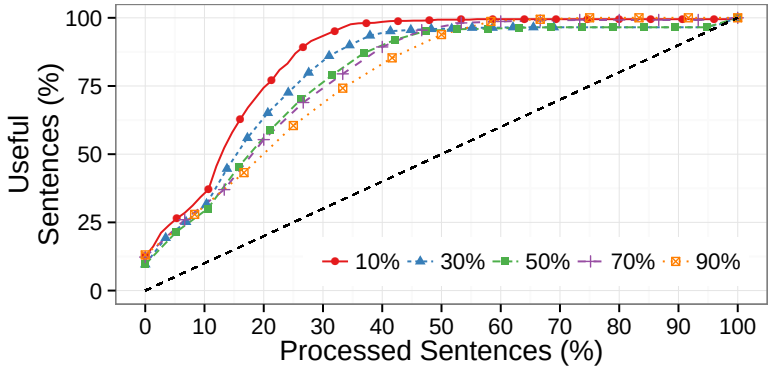


Figure 7.18: Useful sentences recall for Company–Customer for different proportions of useful documents and using the OC extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

Figure 7.19 shows the average recall for the Man Made Disaster–Location relation using the BONG extraction system. Other relations yielded analogous conclusions. As shown, the SVM baseline and our variants of the RSVM-IE approach perform best, as they effectively learn words and phrases that identify—and prioritize—useful sentences with high precision. This is crucial for the BONG extraction system as well as for other extraction systems that exploit primarily the words and phrases in a sentence during tuple extraction. Our approach, on the contrary, does not exploit the words and phrases directly but rather uses a semantic representation of them. Although this representation enables a richer characterization of the novelty and usefulness (see below), it misses shallow features such as words and phrases.

We compared the same techniques above in terms of the unique tuples and attributes that they extract along the extraction process. Figure 7.20 shows the average number of tuples and attributes for the Man Made Disaster–Location relation using the BONG extraction system. We observed similar behavior across relations. There are four broad conclusions that we can draw from examining unique extraction output recall. First, the baseline approaches outperform our GOMP-IE approach during early stages of the extraction process. This correlates with our results above for useful sentence recall. Second, the unique extraction output recall gap is considerably smaller than that of useful sentence recall. For example, the recall of the Man Made Disaster attribute after processing 5% of the sentences is 22% for GOMP-IE and 48% and 57% for the MMR-S and Adap-S variants

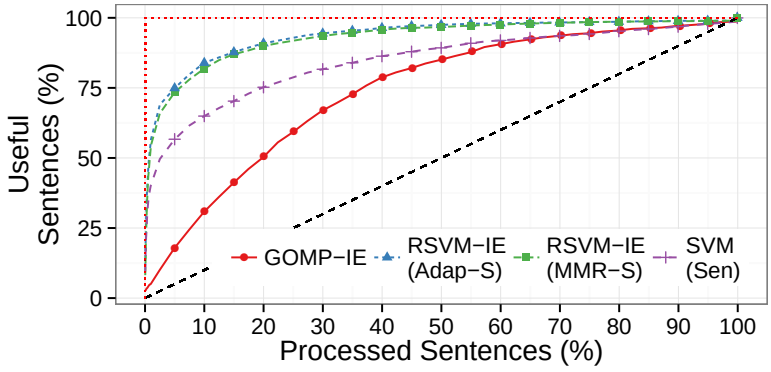


Figure 7.19: Useful sentences recall for Man Made Disaster–Location for different ranking techniques and using the BONG extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

of RSVM-IE, respectively (26% and 35% recall gap); for the same number of processed sentences, the useful sentence recall is 17% for GOMP-IE and 67% and 75% for the MMR-S and Adap-S variants of RSVM-IE, respectively (50% and 58% recall gap). This suggests that GOMP-IE prioritizes novel sentences more effectively than other approaches. This also suggests that by improving the sentence scoring function, which is orthogonal to the ranking approach, we would directly improve the diversity of the extraction process. Third, our approach manages to collect higher fractions of unique tuples and attributes than those of our MMR-S variant of RSVM-IE during late stages of the extraction process. Interestingly, this occurs even when the useful sentence recall for RSVM-IE (MMR-S) is higher than that of GOMP-IE. GOMP-IE also exhibits comparable unique extraction output recall to that of RSVM-IE (Adap-S), even when RSVM-IE (Adap-S) has processed considerably more useful sentences. Fourth, we observe that all recall curves for GOMP-IE exhibit similar values along the extraction process. This indicates that our GOMP-IE approach effectively covers different aspects of the extraction task. The baseline techniques, on the contrary, exhibit significantly different recall curves for their attributes and tuples. For the Man Made Disaster–Location relation, in particular, the recall curve for the Location attribute is higher than that of the Man Made Disaster attribute during early stages of the extraction process. This suggests that the first tuples extracted by using RSVM-IE are heavily biased towards a particular group of useful sentences, since they are predominantly about a small

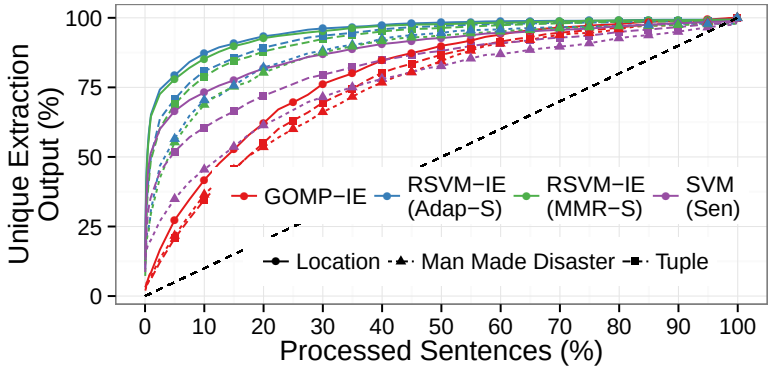


Figure 7.20: Unique extraction output recall for Man Made Disaster–Location for different ranking techniques and using the BONG extraction system.

number of man-made disasters.

Our final analysis involves empirically evaluating the efficiency of our GOMP-IE by measuring the time—including both ranking and extraction time—that each technique requires to achieve different recall values. Figure 7.21 shows the results for the Man Made Disaster–Location relation using the BONG extraction system. Other relations yielded similar results. The processing time per sentence that we measured for BONG is on average 0.139 seconds for useful sentences and 0.0714 seconds for useless sentences. As shown, the sentence ranking approaches that we evaluate produce more efficient executions than simply running the information extraction system over the entire documents. In particular, the techniques that exhibit the best recall in our analysis above, also exhibit the best efficiency.

Overall, our experiments show that GOMP-IE effectively prioritizes useful and novel sentences. During early stages of the process, however, the baseline sentence filtering and ranking approaches, namely, SVM and RSVM-IE, perform substantially better than GOMP-IE. As discussed, these techniques learn a small set of words and phrases that are mentioned in a large portion of the useful sentences. Despite this, GOMP-IE manages to collect a more diverse and balanced set of tuples and attributes than those of the baseline techniques later in the process. We also evaluated GOMP as a (sentence ranking) building block of a system for efficient information extraction, and showed considerable efficiency improvements over systems with distinct recall and precision requirements. Additionally, we evaluated the efficiency of GOMP-IE, and confirmed that the overall extraction time of GOMP-IE—as

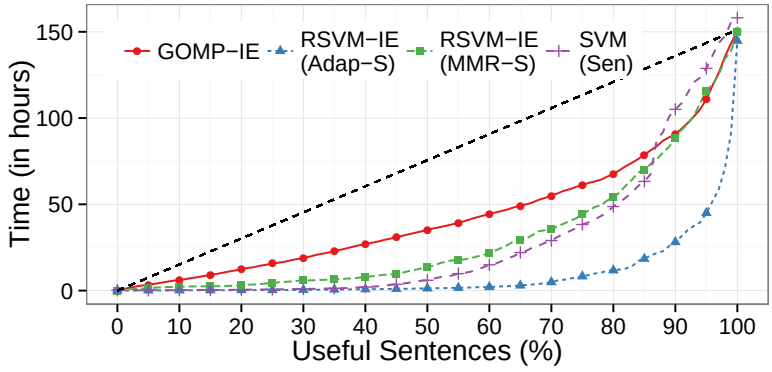


Figure 7.21: CPU time to obtain a target recall value for Man Made Disaster–Location for different ranking techniques and using the BONG extraction system. For reference, we include perfect and random sentence order (see red dotted line and black dashed line, respectively).

well as of the baselines—remains below that of processing all sentences directly.

7.5 Conclusions

In this chapter, we presented a principled, efficient sentence ranking approach for information extraction. Our approach exploits a forward greedy sparse group selection strategy [LSA09] to identify the (rare) useful sentences from a set of documents. Our approach models each sentence as a group of n-grams and iteratively selects the sentence that best explains a carefully designed representation of the extraction task at hand. For usefulness, we build this representation of the extraction task gradually, as the extraction process progresses, to capture all relevant aspects of the task. For novelty, during sentence ranking our approach updates this representation to account for the relevant aspects of the extraction task that have been already explained by other previously selected sentences. Our approach thus manages to prioritize sentences that lead to the extraction of unseen tuples. Furthermore, our approach is flexible, as it enables trading relevance for novelty in a robust manner and, hence, suits different requirements of downstream applications. Our experiments showed the merits and limitations of all relevant building blocks in our approach for both usefulness and novelty and, more importantly, showed the significant efficiency improvements that are enabled by effectively prioritizing sentences. However, the (strong) baseline strategies that

we evaluated performed substantially better than our approach during early stages of the extraction process. These baseline strategies effectively identified a few highly discriminative words and phrases, which lead to processing a high fraction of the useful sentences early in the process. We also showed that our approach improves the efficiency of the extraction process over document sets with different usefulness proportions.

The main contribution of this chapter is our new approach for ranking novel useful sentences for an extraction task of choice. The key building blocks of our approach, namely, sentence representation, useful information representation, and sentence scoring, are well defined and can be adapted to support different representations or scoring approaches. Moreover, our approach can handle text fragments beyond single sentences, such as sets of sentences or paragraphs. Despite being outperformed by baseline strategies that operate over shallow features (e.g., words and phrases), our empirical evaluation has shown the merits and limitations of all relevant building blocks in our approach. These results have shown that improving the sentence scoring technique, which now relies on all words in a sentence, may enhance the performance of our approach considerably. Overall, we showed that prioritizing the extraction effort by focusing on useful sentences yields highly efficient extraction executions. Together with our collection and document ranking techniques (Chapters 5 and 6, respectively), we can efficiently process large volumes of text, which is crucial for the scalability of information extraction.

Chapter 8

Related Work

We surveyed several applications of large-scale information extraction and described related efforts on the implementation of information extraction systems in Chapters 2 and 3, respectively. This chapter now reviews additional literature that is relevant to this dissertation. Section 8.1 outlines related work on document sampling, the problem that we addressed in Chapter 4 for our information extraction setting. Section 8.2 summarizes research on text collection selection, a problem closely related to our work in Chapter 5. Section 8.3 describes prior work on optimization of the information extraction process over text collections, the problem that we covered in Chapters 6 and 7. Finally, Section 8.4 describes general work on Web-scale information extraction, a problem that touches all key aspects of this dissertation.

8.1 Text Document Sampling

Building document samples from text collections has been widely studied, since many important applications (e.g., resource selection in distributed information retrieval [SS11], data analytics [BYG11; ZZD11; ZZD13], and information extraction [AG03; BLNP11a]) require a small, representative summary of the collections over which they operate. In particular, most research efforts on document sampling from text collections have focused on building random samples, as it is well known that the characteristics of a population can often be effectively determined by such samples. We now focus on query-based document sampling;

then, we describe general work on document sampling.

Early approaches for query-based document sampling (e.g., [CC01]) aimed at efficiently collecting representative documents with certain allowed bias—intrinsic to the querying process—in the sample. The approach in [CC01] starts by issuing random words (e.g., obtained from a dictionary) as text queries and then words obtained from the retrieved documents. Despite their bias, the document samples collected following this approach proved very effective for many applications. Unfortunately, this bias is often unacceptable for other applications such as estimating the size of a collection, because it may lead to innacurate estimations. Recent approaches (e.g., [BYG08; ZZD11; ZZD13]) have addressed this problem, in that they aim at collecting truly random document samples from a text collection. The approaches in [BYG08] and [ZZD11] use a large pool of queries (e.g., all n-grams in an external text collection) that together potentially reach all documents of interest in the collections and that is built once and for all. The sampling process consists of: (i) picking a query from the query pool and issuing it to the collection; and (ii) picking a document at random from the returned set of documents and "accepting" (or "rejecting") the document with a certain non-zero probability (e.g., based on the number of terms that the document includes). The approach in [ZZD13], on the other hand, and similarly to the work in [CC01], generates the queries to be issued (see Item (i) above) "on the fly," as it retrieves documents from the collection in a random-walk fashion. Unfortunately, to effectively represent the (rather rare) useful documents in a collection, these approaches would require issuing an exorbitant number of queries. For a given (sub)population of interest (e.g., documents about sports), the approach in [ZZD11] proposes identifying queries that are positively correlated with this population (e.g., query [golf]), a proposition similar in spirit to that of our document sampling approaches in Chapter 4. In turn, [ZZD11] stratifies the sampling process over correlated and uncorrelated queries. Unfortunately, this approach still requires issuing a large number of queries.

More generally, stratified sampling [SSW03] is often used to represent all relevant subpopulations that random sampling would be unable to cover sufficiently. Stratified sampling [SSW03] separates the subpopulations of interest into non-overlapping strata that can be in turn sampled independently from one another. Existing approaches for efficiently run-

ning an information extraction system over a large text collection (e.g., QXtract [AG03], FactCrawl [BLNP11a], PRDualRank [FC11]) often require such stratification to learn discriminative queries that retrieve useful documents: One stratum consists of useful documents, which are collected with high-precision queries (see Section 4.1), whereas the other stratum consists of (rather frequent) useless documents, which we can obtain from a random sample (e.g., by using [CC01], as suggested in [AG03]). A more fine-grained version of stratified document sampling over fully-accessible collections, and for the related problem of optimizing the extraction process (see Section 8.3), is performed in [SGG13]: Here, each stratum includes documents reached by the same retrieval strategy. In Chapter 4, we extensively studied different document sampling techniques for efficiently collecting useful documents for an extraction task of choice. Our techniques produced better-quality samples than those with sampling approaches adopted in the literature. Furthermore, our document samples proved useful for the optimization problems that we tackled along this dissertation (Chapters 5 through 7).

8.2 Text Collection Selection

One of the most crucial tasks in distributed information retrieval is that of text collection selection, or resource selection. Given a user query, find and rank the text collections that are topically relevant to the query (e.g., because they include a substantial number of documents that match the query) without interacting with the collection directly. This task is critical for two main reasons. First, many times only a few collections in the distributed environment include documents that are topically relevant to the query. Second, querying all collections is a very expensive proposition.

Text collection selection techniques are often classified into three broad categories:[1] (i) lexicon-based (e.g., GlOSS [GGMT99], CORI [CLC95; XC98], LM [XC99; SJCO02], and CVV [YL97]), which treat collections as bags of words or n-grams and rely on information retrieval scoring metrics (e.g., cosine similarity) to rank the collections; (ii) surrogate-based (e.g., ReDDE [SC03] and its variants UUM [SC04] and RUM [SC05], CRCS [Sho07], SUSHI

[1] For a detailed survey on text collection selection, see [SS11].

[TS09], and DTF [Fuh96; Fuh99]), which exploit the contents and topical relevance of documents in samples, or surrogates, collected from the collections; and (iii) classification- and clustering-based (e.g., MRDD and QC [VGJL95] and QProber [IG08]), where the task is regarded as either a classification task or a clustering task. In Chapter 5, we evaluated an adaptation of ReDDE for our information extraction scenario. Our experimental results exhibited the limitations of classical resource selection approaches in our information extraction setting. More importantly, our estimator-based approaches (see Chapter 5) exhibited the best performance overall for our text collection ranking task.

8.3 Information Extraction Process Optimization

When running an information extraction system over a large text collection, there are many decisions (e.g., the document retrieval strategy or the order of the operators in the information extraction system) that have a substantial impact on both "completeness" (e.g., in terms of recall) and efficiency (e.g., in terms of running time) of the extraction process. To support these decisions, and to find the best execution plan for a certain objective function (e.g., extract t tuples as fast as possible), several cost-based optimization approaches have been proposed (e.g., [IAJG07; JI09; SDNR07; SGG13]). Most approaches provide for decisions related to the data to process (e.g., the set of documents on which to run the extraction system) and to the extraction system (e.g., which operators to run first or what parameters to use for certain internal operations).

Optimization approaches typically first enumerate different extraction execution plans to, finally, choose the best plan for a given objective function. Some of these approaches, namely, CIMPLE [SDNR07] and SystemT [CKL+10], produce execution plans with identical extraction output; others, namely, SQoUT [JI09] and Holistic-MAP [SGG13], evaluate plans with different extraction output as well. These approaches evaluate all execution plans, or a meaningful subset thereof, using cost models and determine, for instance, the number of documents that need to be processed to reach a certain recall. The decision on which execution plan to adopt is based on the informed output of the cost models.

Importantly, the execution plan does not necessarily have to be generated once and for

all. In fact, approaches such as [IAJG07] propose revising earlier decisions as the extraction process progresses and they collect more relevant information about the actual performance of the execution plan. For instance, the chosen execution plan may be unable to reach a desired recall value because it retrieved—and processed—insufficiently many documents, and this approach may switch to scanning the collection to overcome this shortcoming.

The approaches that we have described in Chapters 5 through 7 are orthogonal to the optimization techniques discussed above. Specifically, our approaches become new choices for the optimizers at the time of enumerating the execution plans and evaluating the cost models over them. However, integrating our approaches into these optimizers has many associated challenges (e.g., cost models should now account for our prioritized execution). We characterize these issues further in our future work discussion (Chapter 10).

8.4 Web-Scale Information Extraction

Multiple efforts have addressed the problem of extracting entities and relations from the Web. The Read The Web research project developed NELL [CBK+10; CBW+10; MHM11], or Never-Ending-Language-Learning system, a computer system that aims to learn new entities and relations continuously. NELL relies on ensemble methods and corrects itself as it better understands natural language. Another well-known research project is the Open Information Extraction project, which developed TextRunner [BCS+07; YCB+07] and ReVerb [FSE11], two systems that aim to extract all relations between pairs of noun phrases. ReVerb, which is now the core of the Open Information Project, imposes certain constrains on the relations (e.g., relations between noun phrases need to be verb phrases that end with prepositions). Unfortunately, open information extraction systems often produce relatively noisy output, which complicates reasoning over the extracted data.

More recently, the DeepDive project [NZRS12b] has focused on producing high-confidence extraction output and on other crucial challenges of large-scale information extraction, such as dealing with conflicting evidence [FSYB11]. Unlike the projects above, DeepDive performs deep natural language processing over text to obtain rich linguistic features (e.g., named entity tags or dependency paths). These features are in turn used for statistical

learning and inference in a wide variety of text-centric tasks, including include relation extraction. DeepDive runs on a carefully designed distributed environment that can process 500 million English Web documents in one week. We argue that systems such as DeepDive, which require running computationally expensive tasks over the text, can largely benefit from the techniques developed in this dissertation. We discuss this issue further in our future work section (Chapter 10).

Structured information can also be obtained from sources other than natural text alone. Several efforts (e.g., [CHW+08; LSC10; VHM+11]) have focused on the extraction of entities and relations from the HTML tables on the Web, which are in the order of millions. Along the same lines, YAGO [SKW07; HSBW13], DBpedia [ABK+07], and Freebase [BEP+08] extract entities and relations from semi-structured sources, such as the Wikipedia infoboxes. These approaches leverage domain-specific rules rather than full-fledged entity and relation extraction systems. Unfortunately, these systems can only extract a limited set of entities and relations, specifically, those for which HTML tables and infoboxes exist. This is an important limitation, as recent research efforts have identified a large variety of entities and relations that would be highly valuable in Web search and that are not included in these types of information sources [GHW+14].

Beyond the information sources discussed above, social media provides unique opportunities for information extraction. Social media many times provides more up to date information than do conventional sources of information (e.g., online news), which makes social media particularly attractive in the information extraction setting. For example, during the 2011 earthquakes in Japan and the Arab Spring, social media sites such as Twitter were more up to date than news websites [PY13]. However, information extraction from social media is challenging for three main reasons: (i) social media documents are typically very short (e.g., Twitter posts are only 140 characters long); (ii) text is often noisy and informal, with non-standard abbreviations and lack of punctuation and capitalization; and (iii) social media documents are often unreliable. Resorting to traditional information extraction strategies thus result in significantly degraded performance [PY13].

Information extraction over social media has mainly focused on event extraction, often regarded a subtask of relation extraction. In this setting, though, multiple social media doc-

uments are required during the extraction process. The approach in [SOM10], for instance, classifies tweets on earthquakes and typhoons and uses a probabilistic spatio-temporal model to determine the trajectory and epicenter of the disaster. Other approaches (e.g., [VHSP10]) evaluate tweets during certain events (e.g., a natural disaster) and identify structured information (e.g., weather, flood level, road conditions) that should be automatically extracted—with information extraction systems—during similar events in the future. An extensive body of work that is complementary to these approaches is developed in [BNG09; BNG11; BNG10]. Here social media documents that cover the same events are clustered, to form richer representations of the events that can largely benefit downstream applications. We believe that the techniques developed in this dissertation will play a key role in deploying and scaling the extraction process over social media documents. In particular, our relation extraction toolkit (Chapter 3) and our sentence ranking approach (Chapter 7) involve several building blocks that can potentially be exploited to tackle the intrinsic challenges of extracting structured information from social media documents.

Chapter 9

Conclusions

In this dissertation, we described key building blocks for supporting large-scale information extraction and presented techniques for scaling the extraction of structured relations to large text collections. Next, we summarize our main contributions.

Toolkit for Building Relation Extraction Systems: In Chapter 3, we studied the development, deployment, and evaluation of relation extraction systems. We characterized the limitations of existing toolkits for this task and introduced REEL, an open-source framework to easily develop and evaluate relation extraction systems. Beyond addressing the limitations of existing toolkits, REEL promotes the integration of other, long-established software libraries for all building blocks in relation extraction. Moreover, REEL effectively addresses the complex requirements of relation extraction and helps developers and researchers produce simple and easy-to-understand source code for their relation extraction systems (see Section 3.5). We believe that these crucial characteristics of REEL will foster research on relation extraction, and we hope they will prove useful to the research community at large. We have made REEL publicly available as open source under the General Public License Version 3 (GPLv3) license, at `http://reel.cs.columbia.edu/`.

Study of Document Sampling Strategies for Information Extraction: In Chapter 4, we studied the problem of query-based document sample generation for information extraction. We identified the key aspects of document sampling for this task and considered their implications along two crucial dimensions of the sampling process, namely, the quality and efficiency of the sampling process. We performed a thorough, large-scale experimental

evaluation over realistic Web collections. We showed that, by carefully choosing the different sampling components, we can obtain sampling executions that are several times more efficient—and with samples of significantly better quality—than those following approaches adopted in the literature. Altogether, our study and experimental evaluation provide a roadmap for addressing this critically important building block for efficient, scalable information extraction.

Methods for Ranking Text Collections for Information Extraction: In Chapter 5, we introduced and addressed the problem of ranking text collections for an information extraction task, to prioritize the extraction effort by focusing on collections with substantial numbers of useful documents for the extraction task. Given a set of text collections, our goal was to estimate the number of useful documents for a given extraction task in each collection to, in turn, rank them according to this estimated value. We cast the problem of estimating the number of useful documents for a task as an instance of the generic problem of estimating a "property" of interest for a collection. This related problem has been extensively studied, and their methods are often classified in three broad classes, namely, surrogate-based, query pool-based, and query pool-free methods. We studied both (adaptations of) state-of-the-art methods across these families as well as information extraction-specific approaches. Our extensive experimental evaluation over realistic Web collections focused on the quality of the ranking and on the efficiency gains achieved with the prioritized execution obtained thereafter. Overall, Chapter 5 showed highly-efficient and scalable executions for the deployment of extraction tasks at scale.

Techniques for Ranking Text Documents for Information Extraction: In Chapter 6, we addressed the problem of ranking documents in text collections for an information extraction task, to prioritize the extraction effort by focusing on documents that are likely to produce tuples for the task at hand. We presented a document ranking approach that relies on learning-to-rank techniques to determine the fine-grained characteristics of useful documents for an extraction task of choice. Furthermore, our approach revises the (learned) ranking decisions periodically as the extraction process progresses and new characteristics of the useful documents are revealed. As crucial building blocks for the efficiency of our techniques, we incorporated online learning algorithms and in-training feature selection. As

a result, our experiments showed that our approach exhibits higher recall and precision than those of state-of-the-art approaches, while keeping the ranking overhead to reasonable levels. Overall, the document ranking approach proposed in Chapter 6 yielded efficient executions even for inexpensive extraction tasks and is, hence, a substantial step towards scalable information extraction.

Approach for Ranking Sentences for Information Extraction: In Chapter 7, we addressed the problem of ranking sentences in a set of documents for an information extraction task, to prioritize the extraction effort by focusing on documents that are likely to produce unseen, novel tuples for the task at hand. We presented a sentence ranking approach that exploits a forward greedy sparse group selection solution to characterize usefulness and novelty of sentences in a robust manner. Our approach models each sentence as a group of words and represents each word in a semantically rich manner using so-called distributed word vectors. To identify useful sentences, our approach correlates sentences to a carefully designed representation of the extraction task that is built gradually, as the extraction process progresses. To identify novel sentences, our approach updates the representation of the extraction task so that it only includes the aspects that remain to be explained. Importantly, our approach allows for trading relevance and novelty in a robust manner to suit different requirements of downstream applications. Our experimental evaluation highlighted the importance of sentence ranking, as well as the merits and limitations of the key building blocks in our approach. In particular, we showed that our baseline strategy, namely, an adaptation for sentence ranking of our document ranking technique in Chapter 6, considerably outperforms our proposed approach during early stages of the extraction process. Our analysis highlighted the performance-critical building blocks of our approach and showed opportunities for further improvement. Overall, our approach yielded efficient executions for a broad range of extraction tasks and over sets of documents with different proportions of useful documents. This renders our approach valuable for further improving the efficiency and scalability of the extraction process.

In summary, in this dissertation we addressed critically important building blocks for improving the efficiency and scalability of information extraction over large text collections. During the first part of this dissertation, we presented a toolkit to easily deploy full-fledged

relation extraction systems and proposed fully automatic document sampling techniques for information extraction: These contributions enabled the deployment of information extraction tasks at scale. During the second part of this dissertation, we proposed approaches to improve the efficiency and scalability of information extraction systems over large text collections. Our approaches prioritize the extraction effort by focusing on the useful collections, the useful documents in these collections, and the useful sentences within these documents. We have supported our conclusions with robust evaluations that considered realistic Web collections as well as extraction tasks of distinct characteristics. We have also made REEL, our toolkit to support the development, deployment, and evaluation of relation extraction systems, available. We hope that REEL as well as the contributions of this dissertation will prove useful to the research community.

Chapter 10

Future Work

Our work suggests interesting directions for future research, which we outline below. As we will see, some directions are immediate extensions of the techniques presented in this dissertation; others convey long-term research endeavors for which our work is naturally valuable.

Exploiting Existing Document Samples for an Extraction Task: In Chapter 4, we discussed the problem of document sampling generation for information extraction, namely, to gather collection-specific, representative samples of useful documents, an often-necessary step for efficient information extraction. As discussed, document sampling techniques need to issue a small number of queries and process a small number of documents, for efficiency. Because of this, document samples many times miss relevant groups of useful documents and, in effect, the overall performance of the extraction process may suffer. Interestingly, Web-accessible text collections often share aspects such as their language and contents [IG02] that are essential throughout the extraction process, as argued throughout this dissertation. An interesting direction for future work would thus be to exploit document samples of other collections to: (i) enrich often-incomplete document samples or, alternatively, (ii) "skip" the sample generation step on certain text collections altogether and, instead, use available document samples for the task at hand.

A similar problem to that of Item (i) above is studied in [IG04], for the related problem of resource selection in distributed information retrieval. Here, collection summaries suffer from a sparse-data problem, in that they often miss many words that appear only

in a few documents. To alleviate this problem, the authors in [IG04] propose combining content summaries from collections categorized under similar topic categories, adopting a shrinkage-based approach from hierarchical document classification [MRMN98]. There are, however, fundamental differences that make this approach not directly applicable to our information extraction setting (e.g., the combination of samples should be done at a relation-specific level, rather than at the collection level), and further research is needed. Likewise, the problem in Item (ii) can be seen as an instance of the generic problem of domain adaptation [DM06], where training data—that is labeled for a given task—and testing data—that is unlabeled—are not necessarily drawn from the same underlying distribution (e.g., because they were obtained from different text collections). This (related) problem has been extensively studied for different natural language processing tasks and, in effect, lays the groundwork for exploring novel solutions in our information extraction setting.

Improving Query Generation Strategies: To identify potentially useful documents for an extraction task of interest, the approaches proposed in this dissertation require issuing learned, extraction-specific queries to the collections. We have thoroughly studied a wide variety of query generation techniques that produce simple text queries (e.g., single-word, phrase, or keyword-combination queries) and that we have used across all extraction tasks and systems indistinguishably (see Chapter 4). A promising direction for future work is thus to improve the query generation step—and potentially the overall extraction process—by (i) exploring other, more expressive query formulations and (ii) choosing query generation strategies according to the characteristics of the extraction tasks and systems, as we discuss next.

Many extraction tasks can potentially benefit from more expressive queries than those studied in this dissertation, provided the document retrieval system in the collections supports such queries. For example, we could improve the precision of the queries obtained for our *Occurs-in* relation by allowing Boolean queries such as [richter AND NOT gerhard], to avoid retrieving documents about Gerhard Richter. Alternatively, we could improve recall by allowing Boolean disjunctive queries such as [earthquake OR tremor OR temblor OR quake] that include synonyms of observed words and phrases. Note that such disjunctive queries potentially improve the efficiency of the extraction process, since the number of

issued queries may decrease.

Beyond improving the query generation strategies in isolation, we can also benefit from studying their interaction with the characteristics of the information extraction systems. For example, a query generation strategy that learns high-precision queries may be desirable for a time-consuming extraction system (e.g., extraction systems that require human input) over another strategy that produces high-recall queries. In this case, the extraction system would only need to run over a very small, potentially useful set of documents. Likewise, a query generation strategy that produces high-recall queries may be desirable for fast extraction systems (e.g., an extraction system based on regular expressions) over another strategy that produces high-precision queries. In this case, the extraction process would remain largely unaffected by the number of documents to process.

Ranking Collections, Documents, and Sentences Along Other Dimensions: In this dissertation, we promoted ranking approaches for collections, documents, and sentences that are driven by usefulness. In Chapter 7, we also explored prioritizing novel sentences (i.e., sentences that produce tuples that have not been seen along the extraction process), to avoid (unnecessarily) running the information extraction system at hand over sentences that produce already seen tuples.

In future work, novelty could also be considered for documents and collections. In particular, we could in principle adapt our approach in Chapter 7 to operate over documents; however, the efficiency of the overall extraction would suffer, because documents would require prohibitively large representations. Instead, we could build on approaches proposed for the related problem of retrieving novel and diverse documents in Web search (e.g., [SCAC14]). Unlike in Web search, though, we should prioritize useful documents with novel tuples rather than documents that are topically relevant to queries that carry multiple interpretations. For collections, we could rely on the intuition that collections that cover different topics may lead to the extraction of largely disjoint sets of tuples, and prioritize the most useful collections for each topic. However, within certain topics (e.g., news, business, sports) some collections may still lead to very different sets of tuples (e.g., because they belong to different geographical areas). Extending our idea of ranking to the novelty of the extraction output is thus an interesting direction for future work.

Beyond novelty and usefulness, several other dimensions are also worth exploring:

- *Ranking for Diversity:* As another ranking dimension worth exploring, we identify diversity of the extraction output, a dimension that is related to novelty but involves different objective functions [CKC+08; VC11]: When ranking for diversity, the goal will be to obtain extraction outputs that differ from one another as a group, and not just with respect to previously seen outputs, as argued for novelty. This problem is related to retrieving diverse and novel documents in information retrieval [DP10; AGHI09], and their adaptation to the information extraction setting poses an interesting and challenging proposition.

- *Ranking for Quality:* Another possible dimension for ranking involves considering the quality of the extraction output. In fact, there may be collections, documents, and sentences that are deemed as relevant but lead to the extraction of low-quality tuples, which subsequent applications (e.g., see Section 2.4) may disregard entirely. The approaches that we have proposed in this dissertation can be adapted to effectively account for the quality of the extraction output: (i) for collections, we could incorporate the quality of the extraction output in the target measure f (e.g., as a function of the confidence scores of the tuples extracted from a document), as suggested in [ZZD13] and [BYG11]; (ii) for documents, we could modify our "binary" interpretation of useful documents during learning (i.e., that a document is useful if it produces a tuple for the relation of choice and useless otherwise) and characterize some useful documents as more useful than others (e.g., if their tuples exhibit different confidence scores); (iii) for sentences, we could assign weights to the different words and phrases in the target matrix that account for the quality of the associated tuples (e.g., as a function of the confidence score of the tuples mentioned within the context of these words and phrases). Integrating these new ranking dimensions is an interesting proposition, as it would combine two of the most important characteristics of the extraction output, namely, volume and quality.

- *Ranking for Efficiency:* Throughout this dissertation we have addressed the efficiency of the extraction process by prioritizing useful collections, documents, and sentences.

Although our usefulness-driven approach has been shown effective, there are other fine-grained aspects of the extraction process that we have not modeled and that directly affect the efficiency of the extraction process. As a notable example of such aspects, consider the average length of sentences and documents in a collection: Many natural language processing tasks (e.g., dependency parsing or named entity tagging, which often are essential building blocks in information extraction systems, as discussed) tend to be slow over lengthy sentences and documents [FKM08]. However, as argued in Chapter 7, long documents and sentences, which are many times prioritized over shorter documents and sentences, are not necessarily more likely to be useful. Accounting for the length of documents and sentences, as well as other efficiency-related aspects, to further improve the efficiency of the extraction process, is therefore an interesting direction of future work.

Along the dimensions discussed above, many variations are possible. For instance, when ranking for novelty and diversity, we could either adopt a rather simple approach (e.g., one that considers only the words and phrases in the text) or explore complex similarity measures across the text (e.g., one that exploits the topics covered in the text or metadata, such as dates or geographical location). Importantly, our work is complementary to these new ranking dimensions and their integration poses many interesting challenges.

Exploiting External Structured and Unstructured Resources: The methods developed in this dissertation rely only on text contents to tackle the efficiency and scalability of the extraction process. However, there are many often-curated structured and unstructured resources that can enhance the plain text for multiple applications. For example, and as described in Section 2.2.1, scientific articles typically include metadata (e.g., authors, affiliations, keywords, category), news articles often include a creation date and entities and topics covered, and Wikipedia documents often link to related Wikipedia documents and external sources, which have been proven useful for many tasks (e.g., citation prediction [YHO+11], event tracking [BNG10], and named entity disambiguation [HOD12]). A natural direction of future work would be to exploit this information to improve our ranking decisions for information extraction.

In addition to the efforts described above, and certainly related to our problem of

focus, distant supervision [MBSJ09] exploits the entities and relations in a knowledge base (e.g., Freebase [BEP+08]) as instances of weakly labeled data: The intuition is that any sentence that contains any pair of entities related in the knowledge base is likely to express the relation in some way. Interestingly, knowledge bases many times include much more information about entities and their relations (e.g., attributes, properties, text descriptions) than just semantic collection between them. An interesting direction of future work would be therefore to integrate this semantically richer information in the form of new features into the representation of our data. By using these (richer) features, we could, for example, identify shared characteristics of geographical locations that are prone to natural disasters—and could potentially occur in *Occurs-in* tuples—and "inject" these characteristics in our ranking approaches, even if we have not observed them in the data.

Finally, other approaches for the related problem of resource selection (e.g., [IG08]) have exploited the categorization of collections, which is generally expected to remain unchanged over time and thus can be used reliably once obtained. The categorization of collections is often determined by the topics covered in the documents in the collection and is of particular importance in our information extraction setting: Relations tend to be topic-specific, as we discussed in Chapter 2. An interesting direction of future work would thus be to capture the "affinity" between topics and relations, so that we focus on collections and documents that cover the topics that are most related to an extraction task of choice. In this problem, topics could be high-level categories in vertical search (e.g., business, finance, sports), for which a natural extension to our setting would build on the approaches in [IG08], or finer-grain entries in other taxonomies and ontologies (e.g., individual nodes or their properties and attributes). For this last case, a possible approach would be to extend probabilistic approaches for modeling the relations between structured entities and sources (e.g., [PDB13]). In particular, the non-parametric approach in [PDB13], namely, the so-called Indian Buffet Process, exhibits many desirable properties for our settings, because the relations and entities therein are unknown ahead of time and are expected to grow as we consider more text contents.

Adding Conditions to the Information Extraction Task: In this dissertation, we have assumed that all the tuples extracted by the information extraction system of choice

are equally valuable, a reasonable assumption for many of the applications discussed in Section 2.4. However, certain applications (e.g., fact checking on the Web [GKK+13]) may only need a subset of the tuples that a given information extraction system would produce. For example, given an information extraction system of the *Occurs-in* relation, we may be interested in natural disasters that occurred within a certain time period or near some predefined geographical location. For these applications, the completion of the extraction task that we promote in our work would be unnecessary. Earlier efforts (e.g., [JDG08]) have incorporated conditions in the document retrieval strategies adopted to identify the documents to process, but have done so in a relatively ad-hoc manner. Extending our work to effectively support such conditioned extraction tasks to all levels of the data is thus of significant interest.

As a first approximation to the problem outlined above, we could train an information extraction system to only produce tuples that meet the given criteria and directly use the approaches presented in this dissertation. Unfortunately, such an approach would require substantial training effort and would not scale well to the large variations of possible conditions that users and applications may impose to extraction tasks. An alternative approach, and one that would not involve training information extraction systems from scratch, would be to redefine the notion of usefulness, so that only the collections, documents, and sentences that lead to the extraction of tuples that meet the given conditions are deemed as useful. This new notion of usefulness is far from trivial, though, because we may still want to leverage knowledge (e.g., words and phrases) observed within the context of tuples that may not match the given conditions, for recall. Furthermore, and as suggested in [JDG08], we should exploit the information provided in the conditions (e.g., that the affected location in an *Occurs-in* tuple should be "San Francisco"). We believe that recommender systems (e.g., [BOHG13; SK09]) will play an important role in this problem, as we could "recommend" certain words and phrases—as well as other signals—to specific conditions, rather than treating all conditions equally as in [JDG08].

Distributing the Extraction Process: In Chapter 2, we briefly discussed the important role of distributed and parallel environments on information extraction. Importantly, there have been several successful efforts in this direction over the past few years. As a notable

example, the DeepDive project [NZRS12b] has leveraged mature parallel-computing infrastructure (e.g., Hadoop [Whi09] and Condor [TTL05]) to speed up the execution of multiple natural language processing tasks (e.g., named entity recognition, dependency parsing) exhaustively over large corpora, to support knowledge-base construction (see Section 2.4). DeepDive runs these tasks over 500 million documents from ClueWeb09 [Clu09] within a week, using Condor and 100,000 CPU hours.

Despite the dramatic efficiency improvements on large-scale text processing with respect to recent predictions (e.g., [PRH04]), we argue that much more efficient text processing executions are possible. In particular, by integrating ranking approaches into distributed infrastructures we can reduce running time—and the number of CPU hours—significantly. For example, the sentences that include mentions of any two entities from Freebase—and hence may include mentions of relations [AMB14; KAH14]—are fewer than 1% of all sentences in ClueWeb09. By focusing the extraction effort we could process—and obtain the tuples from—the relevant data over the same distributed environment in less than two hours. To successfully deploy ranking-based distributed extraction processes over large text collections, however, further research is needed. We believe that our contributions in this dissertation will play a crucial role in this problem.

Putting It All Together: In this dissertation, we have addressed the efficiency and scalability of the extraction process at different granularities of the data, namely, collections, documents, and sentences, independently from one another. A natural direction of future work involves addressing our problem of focus in a holistic approach, where decisions are made considering all granularity levels of the data simultaneously. The following example illustrates this proposition:

Example 4 *Consider an information extraction task T and two text collections C_1 and C_2 that are deemed as useful for T by the estimators in Chapter 5, with $|\widehat{C_1^{\mathrm{U}}}| > |\widehat{C_2^{\mathrm{U}}}|$ (i.e., db_1 is expected to include more useful documents for T than C_2). Based on this, C_1 would be processed before C_2 in a sequential execution. Consider as well that performing the document ranking approach in Chapter 6 over C_1 and C_2 produces substantially fewer tuples from C_1 than from C_2, for the same number of processed documents and issued queries. In this case, ranking C_2 over C_1 would have yielded a more efficient extraction process than that obtained*

by prioritizing collections and documents independently from one another.

As illustrated in Example 4 above, we can produce more efficient executions by considering all granularities of the data holistically. To address this problem, we need optimizers that efficiently enumerate and evaluate execution plans that span all relevant granularities of the data. Importantly, these execution plans will need to account for two relevant dimensions of the extraction process. First, the execution plans should cover the families of techniques available for processing collections, documents, or sentences. In particular, this is challenging because the cost models on which existing optimizers (e.g., [JI09; IAJG07; SGG13]) rely for plan evaluation are not well suited for neither the ranking approaches that we proposed in this dissertation nor their adaptation throughout the extraction process. Second, the execution plans should also cover the deployment of the extraction process over collections, documents, and sentences. This is particularly challenging because the number of plans may become prohibitively large when we consider, for instance, the order in which we process the collections. Furthermore, we may also want to enumerate and evaluate execution plans that combine documents from different collections, which poses additional challenges.

Beyond efficiency, there are other realistic execution scenarios that would benefit from optimization approaches that holistically consider all granularities of the data. Next, we enumerate some of these scenarios along with examples of building blocks that need to be developed to tackle these scenarios effectively:

- *Recall-based scenario:* Many times there is a desired target number of tuples φ to extract [IAJG07]. In this scenario, the goal is to identify the fastest execution plan E to extract φ tuples: (1) $Recall(E) \geq \varphi$ and (2) $Time(E) \leq Time(E_i)$ if $Recall(E_i) \geq \varphi$. For this, we will need estimators to predict the time required to obtain recall φ (e.g., based on the number of queries to issue and number of documents and sentences to process) for a given execution plan. In this case, we can produce time estimates for different portions of the execution plan independently to, in turn, aggregate these estimates. This is possible because this recall-based scenario disregards the actual contents of the extraction output.

- *Diversity-based scenario:* Other scenarios involve defining a target number of distinct tuples δ to extract. Here, the goal is to identify the fastest execution plan E to extract δ distinct tuples: (1) $Diversity(E) \geq \delta$ and (2) $Time(E) \leq Time(E_i)$ if $Diversity(E_i) \geq \delta$. For this, we will need estimators to predict the time required to obtain δ distinct tuples (e.g., based on the number of queries to issue and number of documents and sentences to process). Importantly, we need to predict the extraction overlap that exists in the available text. If we can effectively account for this overlap, we will be able to evaluate the plans in a similar fashion to that of the recall-based scenario above.

- *Time-bound scenario:* Some applications often require the most extraction output in a constrained time τ. The goal in this scenario is to obtain the execution plan E that achieves the best tuple recall, tuple diversity, or a combination thereof, given the time constraints: (1) $F(E) \geq F(E_i)$ and (2) $Time(E), Time(E_i) \leq \tau$ for all i, where F is the target function. For this, we will need estimators to predict the value of F (e.g., recall or diversity) obtained after running the extraction process for a certain period of time.

Earlier optimizers (e.g., [IAJG07]) can potentially be adapted to these scenarios above. These optimizers tackle the optimization problem in two distinct ways. On one hand, they adopt a global strategy that chooses the best execution plan once and for all for a given optimization function. On the other hand, rather than optimizing globally, these optimizers adopt a local strategy that partitions the execution plan into smaller optimization stages (e.g., 5% of the allocated time) and successively identify the best execution plan locally for each stage. This local strategy will be particularly useful for the adaptive strategies proposed in this dissertation, since our optimization decisions can be improved as the extraction process progresses.

Bibliography

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of The Semantic Web, Sixth International Semantic Web Conference, Second Asian Semantic Web Conference (ISWC '07 / ASWC '07)*, 2007. (Cited on pages 22 and 208.)

[AC05] Eugene Agichtein and Silviu Cucerzan. Predicting accuracy of extracting information from unstructured text collections. In *Proceedings of the Fourteenth ACM International Conference on Information and Knowledge Management (CIKM '05)*, 2005. (Cited on pages 2, 21, and 94.)

[AG00] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries (DL '00)*, 2000. (Cited on pages 10, 12, and 128.)

[AG03] Eugene Agichtein and Luis Gravano. Querying text databases for efficient information extraction. In *Proceedings of the Nineteenth International Conference on Data Engineering (ICDE '03)*, 2003. (Cited on pages 3, 17, 18, 19, 51, 55, 56, 58, 61, 67, 69, 70, 74, 75, 92, 99, 105, 111, 126, 129, 139, 140, 160, 165, 203, and 205.)

[AGHI09] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proceedings of the Third ACM International*

Conference on Web Search and Data Mining (WSDM '09), 2009. (Cited on pages 161 and 218.)

[Agi05] Eugene Agichtein. Scaling information extraction to large document collections. *IEEE Data Engineering Bulletin*, 28(4):3–10, 2005. (Cited on page 159.)

[AIG03] Eugene Agichtein, Panagiotis G. Ipeirotis, and Luis Gravano. Modeling query-based access to text databases. In *Proceedings of the Sixth International Workshop on the Web and Databases (WebDB '03)*, 2003. (Cited on page 76.)

[AKM13] Alan Akbik, Oresti Konomi, and Michail Melnikov. Propminer: A workflow for interactive information extraction and exploration using dependency trees. In *Proceedings of the 2013 ACL System Demonstrations (ACL '13)*, 2013. (Cited on page 28.)

[ALG01] Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning search engine specific query transformations for question answering. In *Proceedings of the Tenth International Conference on World Wide Web (WWW '10)*, 2001. (Cited on page 22.)

[AMB14] Alan Akbik, Thilo Michael, and Christoph Boden. Exploratory relation extraction in large text corpora. In *Proceedings of the Twenty-fifth International Conference on Computational Linguistics (COLING '14)*, 2014. (Cited on pages 28 and 222.)

[AS12] Rami Al-Rfou and Steven Skiena. Speedread: A fast named entity recognition pipeline. In *Proceedings of the Twenty-fourth International Conference on Computational Linguistics (COLING '12)*, 2012. (Cited on page 13.)

[BBE+03] Jinbo Bi, Kristin Bennett, Mark Embrechts, Curt Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *The*

Journal of Machine Learning Research, 3:1229–1243, 2003. (Cited on pages 168 and 181.)

[BCS+07] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI '00)*, 2007. (Cited on pages 92, 127, 159, and 207.)

[BDB02] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP '02)*, 2002. (Cited on page 22.)

[BEP+08] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM International Conference on Management of Data (SIGMOD '08)*, 2008. (Cited on pages 22, 208, and 220.)

[BF10] Luciano Barbosa and Juliana Freire. Siphoning hidden-web data through keyword-based interfaces. *Journal of Information and Data Management*, 1(1):133–144, 2010. (Cited on page 68.)

[BGD15] Pablo Barrio, Luis Gravano, and Chris Develder. Ranking deep web text collections for scalable information extraction. In *Proceedings of the Twenty-fourth ACM International Conference on Information and Knowledge Management (CIKM '15)*, 2015. (Cited on page 92.)

[BHL11] Christoph Boden, Thomas Haefele, and Alexander Löser. Classification algorithms for Web text filtering. In *First International Workshop on Managing Data Throughout its Lifecycle (DaLi '11)*, 2011. (Cited on pages 159, 181, and 182.)

[Bir06] Steven Bird. NLTK: The natural language toolkit. In *Proceedings of the Twenty-first International Conference on Computational Linguistics*

and Forty-fourth Annual Meeting of the Association for Computational Linguistics (COLING '06), 2006. (Cited on page 29.)

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer-Verlag, 2006. (Cited on page 132.)

[BLNP11a] Christoph Boden, Alexander Löser, Christoph Nagel, and Stephan Pieper. FactCrawl: A fact retrieval framework for full-text indices. In *Proceedings of the Fourteenth International Workshop on the Web and Databases (WebDB '11)*, 2011. (Cited on pages 3, 17, 18, 19, 70, 71, 92, 105, 126, 127, 140, 155, 165, 203, and 205.)

[BLNP11b] Christoph Boden, Alexander Löser, Christoph Nagel, and Stephan Pieper. FactCrawl: A fact retrieval framework for full-text indices. In *Proceedings of the Fourteenth International Workshop on the Web and Databases (WebDB '11)*, 2011. (Cited on page 160.)

[BLNP12] Christoph Boden, Alexander Löser, Christoph Nagel, and Stephan Pieper. Fact-aware document retrieval for information extraction. *Datenbank-Spektrum*, 12(2):89–100, 2012. (Cited on pages 51, 55, 60, and 70.)

[BM05a] Razvan C. Bunescu and Raymond J. Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP '05)*, 2005. (Cited on pages 11, 12, 14, 49, and 179.)

[BM05b] Razvan C. Bunescu and Raymond J. Mooney. Subsequence kernels for relation extraction. In *Proceedings of the Nineteenth International Conference on Neural Information Processing Systems (NIPS '05)*, 2005. (Cited on pages 11, 12, 28, 49, 67, 109, 139, and 179.)

[BNG09] Hila Becker, Mor Naaman, and Luis Gravano. Event identification in social media. In *Proceedings of the Twelfth International Workshop on the Web and Databases (WebDB '09)*, 2009. (Cited on page 209.)

[BNG10] Hila Becker, Mor Naaman, and Luis Gravano. Learning similarity metrics for event identification in social media. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM '10)*, 2010. (Cited on pages 15, 209, and 219.)

[BNG11] Hila Becker, Mor Naaman, and Luis Gravano. Beyond trending topics: Real-world event identification on twitter. In *Proceedings of the Fifth International Conference on Weblogs and Social Media (ICWSM '11)*, 2011. (Cited on page 209.)

[BOHG13] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. (Cited on page 221.)

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the Nineteenth International Conference on Computational Statistics (COMPSTAT '11)*, 2010. (Cited on page 131.)

[Boy82] Bert R. Boyce. Beyond topicality : A two stage view of relevance and the retrieval process. *Information Processing and Management*, 18(3):105–109, 1982. (Cited on page 161.)

[BP06] Razvan Bunescu and Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL '06)*, 2006. (Cited on page 22.)

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. (Cited on page 133.)

[Bri98] Sergey Brin. Extracting patterns and relations from the world wide web. In *Proceedings of the First International Workshop on the Web and Databases (WebDB '98)*, 1998. (Cited on page 128.)

[Bri99] Sergey Brin. Extracting patterns and relations from the world wide web. In *Proceedings of the Second International Workshop on the Web and Databases (WebDB '99)*. 1999. (Cited on page 12.)

[BRMB11] Falk Brauer, Robert Rieger, Adrian Mocan, and Wojciech M. Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *Proceedings of the Twentieth ACM International Conference on Information and Knowledge Management (CIKM '11)*, 2011. (Cited on page 28.)

[BSGG14] Pablo Barrio, Gonçalo Simões, Helena Galhardas, and Luis Gravano. REEL: A relation extraction learning framework. In *Proceedings of the 2014 ACM Joint Conference on Digital Libraries (JCDL '14)*, 2014. (Cited on page 28.)

[BSGG15] Pablo Barrio, Gonçalo Simões, Helena Galhardas, and Luis Gravano. Learning to rank adaptively for scalable information extraction. In *Proceedings of the 2015 International Conference on Extending Database Technology (EDBT '15)*, 2015. (Cited on page 126.)

[BWG+10] Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Olivier Chapelle, and Kilian Weinberger. Learning to rank with (a lot of) word features. *Information Retrieval*, 13(3):291–314, 2010. (Cited on page 131.)

[BYG08] Ziv Bar-Yossef and Maxim Gurevich. Random sampling from a search engine's index. *Journal of the ACM*, 55(5):1–74, 2008. (Cited on pages 53, 96, 102, and 204.)

[BYG11] Ziv Bar-Yossef and Maxim Gurevich. Efficient search engine measurements. *ACM Transactions on the Web*, 5(4):18:1–18:48, 2011. (Cited on pages 16, 94, 95, 99, 101, 102, 105, 107, 112, 203, and 218.)

[CA05] Silviu Cucerzan and Eugene Agichtein. Factoid question answering over unstructured and structured web content. In *Proceedings of the Fourteenth Text Retrieval Conference (TREC-14)*, 2005. (Cited on page 23.)

[CBK+10] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI '10)*, 2010. (Cited on pages 22 and 207.)

[CBW+10] Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM '10)*, 2010. (Cited on page 207.)

[CC01] Jamie Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001. (Cited on pages 53, 109, 111, 112, 204, and 205.)

[CCB95] James P. Callan, William Bruce Croft, and John Broglio. TREC and TIPSTER experiments with INQUERY. *Information Processing and Management*, 31(3):327–343, 1995. (Cited on page 98.)

[CG98] Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the Twenty-first ACM International Conference on Research and Development in Information Retrieval (SIGIR '98)*, 1998. (Cited on pages 161 and 182.)

[CGN05] Niladri Chatterjee, Shailly Goyal, and Anjali Naithani. Resolving pattern ambiguity for english to hindi machine translation using WordNet. In *Workshop on Modern Approaches in Translation Technologies at RANLP (RANLP '05)*, 2005. (Cited on page 22.)

[Chi98] Nancy A. Chinchor. Overview of muc-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998. (Cited on page 10.)

[Cho12] Jinho D. Choi. *Optimization of Natural Language Processing Components for Robustness and Scalability*. PhD thesis, Boulder, CO, USA, 2012. AAI3549172. (Cited on page 13.)

[CHW+08] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. WebTables: Exploring the power of tables on the web. *VLDB Endowment*, 1(1):538–549, 2008. (Cited on page 208.)

[CJTN06] Jinxiu Chen, Donghong Ji, Chew Lim Tan, and Zhengyu Niu. Relation extraction using label propagation based semi-supervised learning. In *Proceedings Twenty-first International Conference on Computational Linguistics and Forty-fourth Annual Meeting of the Association for Computational Linguistics (COLING '06)*, 2006. (Cited on page 129.)

[CK06] Harr Chen and David R. Karger. Less is more: Probabilistic models for retrieving fewer relevant documents. In *Proceedings of the Twenty-ninth ACM International Conference on Research and Development in Information Retrieval (SIGIR '06)*, 2006. (Cited on page 161.)

[CKC+08] Charles L. A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the Thirty-first ACM International Conference on Research and Development in Information Retrieval (SIGIR '08)*, 2008. (Cited on pages 161 and 218.)

[CKL+10] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick R. Reiss, and Shivakumar Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *Proceedings of the Forty-eighth Annual Meeting of the Association for Computational Linguistics (ACL '10)*, 2010. (Cited on page 206.)

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011. (Cited on page 29.)

[CLC95] James P. Callan, Zhihong Lu, and William Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the Eighteenth ACM International Conference on Research and Development in Information Retrieval (SIGIR '95)*, 1995. (Cited on pages 93, 95, and 205.)

[CLTW10] Bo Chen, Wai Lam, Ivor W. Tsang, and Tak-Lam Wong. Location and scatter matching for dataset shift in text mining. In *Proceedings of the Tenth IEEE International Conference on Data Mining (ICDM '10)*, 2010. (Cited on page 75.)

[Clu09] The ClueWeb09 Dataset. http://www.lemurproject.org/clueweb09.php/, 2009. [Online; accessed August 2015]. (Cited on pages 14 and 222.)

[Clu12] The ClueWeb12 Dataset. http://www.lemurproject.org/clueweb12.php/, 2012. [Online; accessed August 2015]. (Cited on page 14.)

[CNN15] Cable News Network. http://www.cnn.com/, 2015. [Online; accessed August 2015]. (Cited on page 14.)

[Coh95] William W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML '98)*, 1995. (Cited on page 70.)

[Coh04] William W. Cohen. Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data. http://minorthird.sourceforge.net, 2004. (Cited on page 29.)

[CS04] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the Forty-second Annual Meeting of the*

Association for Computational Linguistics (ACL '04), 2004. (Cited on pages 11 and 12.)

[Cvi10] Ana Cvitas. Information extraction in business intelligence systems. In *Proceedings of the Thirty-third International Convention on Information and Communication Technology, Electronics, and Microelectronics (MIPRO '10)*, 2010. (Cited on page 23.)

[CWW02] Michael Callaham, Robert L. Wears, and Ellen Weber. Journal prestige, publication bias, and other characteristics associated with citation of published studies in peer-reviewed journals. *Journal of the American Medical Association*, 287(21):2847–2850, 2002. (Cited on page 24.)

[D4D14] D4D Challenge – Senegal. http://www.d4d.orange.com/, 2014. [Online; accessed August 2015]. (Cited on page 14.)

[Dar13] Brittany Darwell. Facebook builds knowledge graph with info modules on community pages. http://www.adweek.com/socialtimes/facebook-builds-knowledge-graph-with-info-modules-on-community-pages, 2013. [Online; accessed August 2015]. (Cited on page 22.)

[DC12] Van Dang and William Bruce Croft. Diversity by proportionality: An election-based approach to search result diversification. In *Proceedings of the Thirty-Fifth ACM International Conference on Research and Development in Information Retrieval (SIGIR '12)*, 2012. (Cited on page 161.)

[DEG+03] Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, Ramanathan Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John A. Tomlin, and Jason Y. Zien. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the Twelfth International Conference on World Wide Web (WWW '03)*, 2003. (Cited on page 20.)

[DGH+14] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang.

Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the Twentieth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD '14)*, 2014. (Cited on page 22.)

[DLT+13] Omkar Deshpande, Digvijay S. Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, maintaining, and using knowledge bases: A report from the trenches. In *Proceedings of the 2013 ACM International Conference on Management of Data (SIGMOD '13)*, 2013. (Cited on page 22.)

[DM06] Hal Daumé, III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26(1):101–126, 2006. (Cited on page 216.)

[DMA97] Geoffrey Davis, Stephane Mallat, and Marco Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997. (Cited on page 164.)

[DMP+] George R. Doddington, Alexis Mitchell, Mark A. Przybocki, Lance A. Ramshaw, Stephanie Strassel, and Ralph M. Weischedel. The automatic content extraction (ACE) program-tasks, data, and evaluation. (Cited on page 10.)

[DP10] Marina Drosou and Evaggelia Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010. (Cited on page 218.)

[dSMSB13] Filipe de Sá Mesquita, Jordan Schmidek, and Denilson Barbosa. Effectiveness and efficiency of open relation extraction. In *Proceedings of the Fourteenth Conference on Empirical Methods in Natural Language Processing (EMNLP '13)*, 2013. (Cited on page 13.)

[Dun93] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993. (Cited on page 70.)

[DW15] Sourav Dutta and Gerhard Weikum. Cross-document co-reference resolution using sample-based clustering with knowledge enrichment. *Transactions of the Association for Computational Linguistics*, 3:15–28, 2015. (Cited on page 13.)

[DYFC09] Ying Ding, Erjia Yan, Arthur Frazho, and James Caverlee. Pagerank for ranking authors in co-citation networks. *Journal of the American Society for Information Science and Technology*, 60(11):2229–2243, 2009. (Cited on page 24.)

[EB07] Asif Ekbal and Sivaji Bandyopadhyay. A hidden markov model based named entity recognition system: Bengali and hindi as case studies. In *Proceedings of the Second International Conference on Pattern Recognition and Machine Intelligence (PReMI '07)*, 2007. (Cited on page 138.)

[EBSW08] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008. (Cited on pages 10 and 22.)

[ECD+05] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005. (Cited on page 22.)

[EGN14] Kathrin Eichler, Aleksandra Gabryszak, and Günter Neumann. An analysis of textual inference in german customer emails. In *Proceedings of the Third Joint Conference on Lexical and Computational Semantics (SEM '14)*, 2014. (Cited on page 14.)

[Els15] Elsevier: Publishing company. http://www.elsevier.com/, 2015. [Online; accessed August 2015]. (Cited on page 14.)

[ESI+12] Edward A. Epstein, Marshall I. Schor, Bhavani Iyer, Adam Lally, Eric W. Brown, and Jaroslaw Cwiklik. Making watson fast. *IBM Journal of Research and Development*, 56(3):15, 2012. (Cited on page 23.)

[Etx12] E-txt2DB: Entity recognition framework. http://web.ist.utl.pt/ist155840/etxt2db/, 2012. [Online; accessed August 2015]. (Cited on pages 33, 67, 110, 139, and 179.)

[FB11] Emilio Ferrara and Robert Baumgartner. Automatic wrapper adaptation by tree edit distance matching. In *Proceedings of the Second International Workshop on Combinations of Intelligent Methods and Applications (CIMA '11)*. 2011. (Cited on pages 65 and 66.)

[FC11] Yuan Fang and Kevin Chen-Chuan Chang. Searching patterns for relation extraction over the web: Rediscovering the pattern-relation duality. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM '11)*, 2011. (Cited on pages 3, 12, 18, 19, 40, 51, 55, 67, 68, 92, 126, 127, 140, 165, and 205.)

[FEM15] Federal Emergency Management Agency. http://www.fema.gov/, 2015. [Online; accessed August 2015]. (Cited on pages 15 and 91.)

[FFGK10] James Fan, David Ferrucci, David Gondek, and Aditya Kalyanpur. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In *Proceedings of the First International Workshop on Formalisms and Methodology for Learning by Reading at NAACL HLT (FAMLBR '10)*, 2010. (Cited on page 22.)

[Fis36] Ronald Aylmer Fisher. *Statistical methods for research workers*. Number 5. Genesis Publishing Pvt Ltd, 1936. (Cited on page 112.)

[FKM08] Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of the Forty-sixth Annual Meeting of the Association for Computational Linguistics (ACL '08)*, 2008. (Cited on page 219.)

[FL04] David Ferrucci and Adam Lally. UIMA: An architectural approach to unstructured information processing in the corporate research environ-

ment. *Natural Language Engineering*, 10(3-4):327–348, 2004. (Cited on page 29.)

[FM09] Jenny Rose Finkel and Christopher D. Manning. Nested named entity recognition. In *Proceedings of the Eighth Conference on Empirical Methods in Natural Language Processing (EMNLP '09)*, 2009. (Cited on page 13.)

[Fod02] Imola Fodor. A survey of dimension reduction techniques. Technical report, 2002. (Cited on page 168.)

[FSE11] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Twelfth Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*, 2011. (Cited on pages 11, 12, 22, 28, and 207.)

[FSYB11] Lujun Fang, Anish Das Sarma, Cong Yu, and Philip Bohannon. REX: Explaining relationships between entity pairs. *VLDB Endowment*, 5(3):241–252, 2011. (Cited on page 207.)

[Fuh96] Norbert Fuhr. Optimum database selection in networked ir. In *Proceedings of the Workshop Networked Information Retrieval at SIGIR (SIGIR '96)*, 1996. (Cited on page 206.)

[Fuh99] Norbert Fuhr. A decision-theoretic approach to database selection in networked ir. *ACM Transactions on Information Systems*, 17(3):229–249, 1999. (Cited on page 206.)

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003. (Cited on pages 7, 125, and 132.)

[Gen15] Genius: Annotate the world. http://genius.com/, 2015. [Online; accessed August 2015]. (Cited on page 25.)

[GGMT99] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. GlOSS: text-source discovery over the internet. *ACM Transactions on Database Systems*, 24(2):229–264, 1999. (Cited on pages 93, 95, and 205.)

[GHW+14] Rahul Gupta, Alon Halevy, Xuezhi Wang, Steven Euijong Whang, and Fei Wu. Biperpedia: An ontology for search applications. *VLDB Endowment*, 7(7):505–516, 2014. (Cited on page 208.)

[GHY02] Ralph Grishman, Silja Huttunen, and Roman Yangarber. Information extraction for enhanced access to disease outbreak reports. *Journal of Biomedical Informatics*, 35(4):236–246, 2002. (Cited on page 159.)

[GIS03] Luis Gravano, Panagiotis G. Ipeirotis, and Mehran Sahami. QProber: A system for automatic classification of hidden-web databases. *ACM Transactions on Information Systems*, 21(1):1–41, 2003. (Cited on pages 64 and 109.)

[GJJM05] Zhou GuoDong, Su Jian, Zhang Jie, and Zhang Min. Exploring various knowledge in relation extraction. In *Proceedings of the Forty-third Annual Meeting on Association for Computational Linguistics (ACL '05)*, 2005. (Cited on pages 11 and 12.)

[GKK+13] François Goasdoué, Konstantinos Karanasos, Yannis Katsis, Julien Leblay, Ioana Manolescu, and Stamatis Zampetakis. Fact checking and analyzing the web. In *Proceedings of the 2013 ACM International Conference on Management of Data (SIGMOD '13)*, 2013. (Cited on page 221.)

[GLM12] Assaf Glazer, Michael Lindenbaum, and Shaul Markovitch. Feature shift detection. In *Proceedings of the Twenty-first International Conference on Pattern Recognition (ICPR '12)*, 2012. (Cited on pages 135 and 141.)

[GLR06] Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of the Eleventh Conference of the European Chap-*

ter of the Association for Computational Linguistics (EACL '06), 2006. (Cited on pages 67, 109, 138, and 179.)

[GM14] Sonal Gupta and Christopher Manning. SPIED: Stanford pattern based information extraction and diagnostics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces (ILLVI '14)*, 2014. (Cited on page 28.)

[Gof64] William Goffman. A searching procedure for information retrieval. *Information Storage and Retrieval*, 2(2):73–78, 1964. (Cited on page 161.)

[GS96] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING '96)*, 1996. (Cited on page 10.)

[GSW05] Jens Graupmann, Ralf Schenkel, and Gerhard Weikum. The SphereSearch engine for unified ranked retrieval of heterogeneous XML and web documents. In *Proceedings of the Thirty-first International Conference on Very Large Databases (VLDB '05)*, 2005. (Cited on page 22.)

[HDA13] Jessica Hullman, Nicholas Diakopoulos, and Eytan Adar. Contextifier: Automatic generation of annotated stock visualizations. In *Proceedings of the Thirty-first Annual Conference on Human Factors in Computing Systems (CHI '13)*, 2013. (Cited on page 25.)

[HDG00] Kevin Humphreys, George Demetriou, and Robert Gaizauskas. Two applications of information extraction to biological science journal articles: enzyme interactions and protein structures. *Pacific Symposium on Biocomputing*, pages 505–516, 2000. (Cited on page 23.)

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. (Cited on pages 29, 37, and 70.)

[HGO00] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, 2000. (Cited on pages 132 and 136.)

[HLN] Warren A. Hunt, Lucian V. Lita, and Eric Nyberg. Gazetteers, WordNet, Encyclopedias, and the Web: Analyzing question answering resources. Technical report. (Cited on page 22.)

[HOD12] Sherzod Hakimov, Salih Atilay Oto, and Erdogan Dogdu. Named entity recognition and disambiguation using linked data and graph-based centrality scoring. In *Proceedings of the Fourth International Workshop on Semantic Web Information Management (SWIM '12)*, 2012. (Cited on pages 15 and 219.)

[HPZC07] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the deep web. *Communications of the ACM*, 50(5):94–101, 2007. (Cited on page 16.)

[HSB+10] Dzung Hong, Luo Si, Paul Bracke, Michael Witt, and Tim Juchcinski. A joint probabilistic classification model for resource selection. In *Proceeding of the Thirty-third ACM International Conference on Research and Development in Information Retrieval (SIGIR '10)*, 2010. (Cited on page 95.)

[HSBW13] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013. (Cited on pages 22 and 208.)

[htm15] Html Cleaner: Transform HTML to well-formed XML. http://htmlcleaner.sourceforge.net/, 2015. [Online; accessed August 2015]. (Cited on page 66.)

[HYBV05] Lynette Hirschman, Alexander Yeh, Christian Blaschke, and Alfonso Valencia. Overview of BioCreAtIvE: Critical assessment of information ex-

traction for biology. *BMC Bioinformatics*, 6(Suppl 1):S1, 2005. (Cited on page 10.)

[HZM09] Junzhou Huang, Tong Zhang, and Dimitris Metaxas. Learning with structured sparsity. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML '09)*, 2009. (Cited on page 162.)

[HZW10] Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. Learning 5000 relational extractors. In *Proceedings of the Forty-eighth Annual Meeting of the Association for Computational Linguistics (ACL '10)*, 2010. (Cited on page 22.)

[IAJG07] Panagiotis G. Ipeirotis, Eugene Agichtein, Pranay Jain, and Luis Gravano. Towards a query optimizer for text-centric tasks. *ACM Transactions on Database Systems*, 32(4):2–47, 2007. (Cited on pages 21, 60, 129, 206, 207, 223, and 224.)

[IBGC+14] Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the Fifteenth Conference on Empirical Methods in Natural Language Processing (EMNLP '14)*, 2014. (Cited on page 23.)

[IC06] José Iria and Fabio Ciravegna. A methodology and tool for representing language resources for information extraction. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC '06)*, 2006. (Cited on page 29.)

[ICW11] ICWSM 2011 Spinn3r Dataset. http://www.icwsm.org/data/, 2011. [Online; accessed August 2015]. (Cited on page 14.)

[IG02] Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the Twenty-eighth International Conference on Very Large Data Bases (VLDB '02)*, 2002. (Cited on page 215.)

[IG04] Panagiotis G. Ipeirotis and Luis Gravano. When one sample is not enough: Improving text database selection using shrinkage. In *Proceedings of the 2004 ACM International Conference on Management of Data (SIGMOD '04)*, 2004. (Cited on pages 215 and 216.)

[IG08] Panagiotis G. Ipeirotis and Luis Gravano. Classification-aware hidden-web text database selection. *ACM Transactions on Information Systems*, 26(2):6:1–6:66, 2008. (Cited on pages 105, 206, and 220.)

[Iri05] José Iria. T-Rex: A flexible relation extraction framework. In *Proceedings of the Eighth Annual Colloquium for the UK Special Interest Group for Computational Linguistics (CLUK '05)*, 2005. (Cited on page 29.)

[IW06] Georgiana Ifrim and Gerhard Weikum. Transductive learning for text classification using explicit knowledge models. In *Proceedings of Tenth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '06)*, 2006. (Cited on page 22.)

[JDG08] Alpa Jain, AnHai Doan, and Luis Gravano. Optimizing SQL queries over text databases. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE '08)*, 2008. (Cited on page 221.)

[JI09] Alpa Jain and Panagiotis G. Ipeirotis. A quality-aware optimizer for information extraction. *ACM Transactions on Database Systems*, 34(1):5:1–5:48, 2009. (Cited on pages 206 and 223.)

[JKR+06] Thathachar S. Jayram, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin*, 29(1):40–48, 2006. (Cited on page 10.)

[Joa98a] Thorsten Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*. 1998. (Cited on page 69.)

[Joa98b] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the Tenth European Conference on Machine Learning (ECML '98)*, 1998. (Cited on pages 40 and 133.)

[Joa03] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Ninth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD '03)*, 2003. (Cited on page 134.)

[JS09] Alpa Jain and Divesh Srivastava. Exploring a few good tuples from text databases. In *Proceedings of the Twenty-fifth International Conference on Data Engineering (ICDE '09)*, 2009. (Cited on pages 2, 21, and 94.)

[KAH14] Johannes Kirschnick, Alan Akbik, and Holmer Hemsen. Freepal: A large collection of deep lexico-syntactic patterns for relation extraction. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC '14)*, 2014. (Cited on page 222.)

[Kam04] Nanda Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the Forty-second Annual Meeting of the Association for Computational Linguistics (ACL '04)*, 2004. (Cited on pages 11 and 12.)

[KKJ+15] Juho Kim, Eun-Young Ko, Jonghyuk Jung, Chang Won Lee, Nam Wook Kim, and Jihee Kim. Factful: Engaging taxpayers in the public discussion of a government budget. In *Proceedings of the Thirty-third Annual Conference on Human Factors in Computing Systems (CHI '15)*, 2015. (Cited on page 25.)

[KKK06] Yuwon Kim, Jinseog Kim, and Yongdai Kim. Blockwise sparse regression. *Statistica Sinica*, 16(2):375–390, 2006. (Cited on page 163.)

[KL51] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951. (Cited on page 69.)

[KM11] Oleksandr Kolomiyets and Marie-Francine Moens. A survey on question answering technology from an information retrieval perspective. *Information Sciences*, 181(24):5412–5434, 2011. (Cited on page 22.)

[Kum02] Vipin Kumar. *Introduction to parallel computing.* Addison-Wesley Longman, 2nd edition, 2002. (Cited on page 20.)

[KV10] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proceedings of the Nineteenth International Conference on World Wide Web (WWW '10)*, 2010. (Cited on page 136.)

[KZ01] Marcin Kaszkiel and Justin Zobel. Effective ranking with arbitrary passages. *Journal of the American Society for Information Science and Technology*, 52(4):344–364, 2001. (Cited on pages 4 and 160.)

[LCY+12] Yunyao Li, Laura Chiticariu, Huahai Yang, Frederick R. Reiss, and Arnaldo Carreno-fuentes. WizIE: A best practices guided development environment for information extraction. In *Proceedings of the 2012 ACL System Demonstrations (ACL '12)*, 2012. (Cited on page 28.)

[LGD15] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. (Cited on page 189.)

[LGG+01] Nicholas M Luscombe, Dov Greenbaum, Mark Gerstein, et al. What is bioinformatics? A proposed definition and overview of the field. *Methods of Information in Medicine*, 40(4):346–358, 2001. (Cited on page 23.)

[lin15] LingPipe: Natural language processing and text analytics. http://alias-i.com/lingpipe/, 2015. [Online; accessed August 2015]. (Cited on pages 29, 71, and 139.)

[Liu09] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. (Cited on page 130.)

[LKT+15] Yunyao Li, Elmer Kim, Marc A. Touchette, Ramiya Venkatachalam, and Hao Wang. Vinery: A visual IDE for information extraction. *VLDB Endowment*, 8(12):1948–1959, 2015. (Cited on page 28.)

[LLYM04] Shuang Liu, Fang Liu, Clement Yu, and Weiyi Meng. An effective approach to document retrieval via utilizing WordNet and recognizing phrases. In *Proceedings of the Twenty-seventh ACM International Conference on Research and Development in Information Retrieval (SIGIR '04)*, 2004. (Cited on page 22.)

[LM14] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning (ICML '14)*, 2014. (Cited on pages 166 and 178.)

[LSA09] Aurelie C. Lozano, Grzegorz Swirszcz, and Naoki Abe. Grouped orthogonal matching pursuit for variable selection and prediction. In *Advances in Ranking Workshop at NIPS (NIPS '09)*, 2009. (Cited on pages 7, 157, 162, 163, 164, and 201.)

[LSA11] Aurelie C. Lozano, Grzegorz Swirszcz, and Naoki Abe. Group orthogonal matching pursuit for logistic regression. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS '11)*, 2011. (Cited on pages 162, 163, 164, 170, and 171.)

[LSC10] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *VLDB Endowment*, 3(1-2):1338–1347, 2010. (Cited on page 208.)

[LSYM01] King-Lup Liu, Adrain Santoso, Clement Yu, and Weiyi Meng. Discovering the representative of a search engine. In *Proceedings of the Tenth ACM International Conference on Information and Knowledge Management (CIKM '01)*, 2001. (Cited on page 95.)

[Luc15] The Lucene Indexer. http://lucene.apache.org/, 2015. [Online; accessed August 2015]. (Cited on pages 109, 127, and 138.)

[LZBB13] Raymond Liaw, Ari Zilnik, Mark Baldwin, and Stephanie Butler. Maater: Crowdsourcing to improve online journalism. In *Proceedings of the Thirty-first Annual Conference on Human Factors in Computing Systems (CHI '13)*, 2013. (Cited on page 25.)

[LZY01] Jianming Li, Lei Zhang, and Yong Yu. Learning to generate semantic annotation for domain specific sentences. In *Proceedings of the K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation (Semannot@K-CAP '01)*, 2001. (Cited on page 20.)

[MAAH09] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Halevy. Harnessing the deep web: Present and future. 2009. (Cited on page 15.)

[MBF+90] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to WordNet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244, 1990. (Cited on page 22.)

[MBGM04] Dunja Mladenic, Janez Brank, Marko Grobelnik, and Natasa Milic-Frayling. Feature selection using linear classifier weights: interaction with classification models. In *Proceedings of the Twenty-seventh ACM International Conference on Research and Development in Information Retrieval (SIGIR '04)*, 2004. (Cited on pages 69 and 110.)

[MBN02] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of the Second IEEE International Conference on Data Mining (ICDM '02)*, 2002. (Cited on page 168.)

[MBSJ09] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the*

Forty-seventh Annual Meeting of the Association for Computational Linguistics and the Fourth International Joint Conference on Natural Language Processing of the AFNLP (IJCNLP '09), 2009. (Cited on page 220.)

[McC02] Andrew K. McCallum. MALLET: A Machine Learning for Language Toolkit. http://mallet.cs.umass.edu, 2002. (Cited on page 29.)

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the 2013 International Conference on Learning Representations (ICLR '13)*, 2013. (Cited on pages 166 and 180.)

[MCSSMTLA13] Heidy M. Marin-Castro, Victor J. Sosa-Sosa, Jose F. Martinez-Trinidad, and Ivan Lopez-Arevalo. Automatic discovery of web query interfaces using machine learning techniques. *Journal of Intelligent Information Systems*, 40(1):85–108, 2013. (Cited on page 65.)

[MFP00] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*, 2000. (Cited on pages 67, 110, 138, 139, and 179.)

[MHM11] Thahir P. Mohamed, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Discovering relations between noun categories. In *Proceedings of the Twelfth Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*, 2011. (Cited on page 207.)

[MIC+15] Kathleen McKeown, Hal Daume III, Snigdha Chaturvedi, John Paparrizos, Kapil Thadani, Pablo Barrio, Or Biran, Suvarna Bothe, Michael Collins, Kenneth Fleischmann, Luis Gravano, Rahul Jha, Ben King, Kevin McInerney, Taesun Moon, Diarmuid O'Seaghdha, Dragomir Radev, Clay Templeton, and Simone Teufel. Predicting impact of scientific con-

cepts using full text features. [to appear]. *Journal of the American Society for Information Science and Technology*, 2015. (Cited on page 24.)

[MJC+07] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR '07)*, 2007. (Cited on page 16.)

[MKK+08] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *VLDB Endowment*, 1(2):1241–1252, 2008. (Cited on pages 15, 16, and 65.)

[ML03] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL '05)*, 2003. (Cited on pages 67, 110, 139, and 179.)

[MNPT02] Bernardo Magnini, Matteo Negri, Roberto Prevete, and Hristo Tanev. Is it the right answer?: Exploiting web redundancy for answer validation. In *Proceedings of the Fortieth Annual Meeting on Association for Computational Linguistics (ACL '02)*, 2002. (Cited on page 22.)

[Moo20] Eliakim Hastings Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395, 1920. (Cited on page 173.)

[Moo11] Raymond Mooney. Aimed: Protein-protein interactions, 2011. [Online; accessed August 2015]. (Cited on page 33.)

[MRMN98] Andrew McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*, 1998. (Cited on page 216.)

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. (Cited on page 137.)

[MSB+12] Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. Open language learning for information extraction. In *Proceedings of the Thirteenth Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12)*, 2012. (Cited on page 22.)

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Twenty-Seventh International Conference on Neural Information Processing Systems (NIPS '13)*, 2013. (Cited on pages 167 and 180.)

[MVDGB08] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society*, 70:53–71, 2008. (Cited on page 163.)

[MZ93] Stéphane G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993. (Cited on page 165.)

[NTW11] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM '11)*, 2011. (Cited on page 22.)

[Nut15] The Nutch Crawler: Highly extensible, highly scalable Web crawler. http://nutch.apache.org/, 2015. [Online; accessed August 2015]. (Cited on page 109.)

[NVB01] Claire Nedellec, Mohamed Ould Abdel Vetah, and Philippe Bessières. Sentence filtering for information extraction in genomics, a classification

problem. In *Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Discovery (ECMLPKDD '01)*, 2001. (Cited on pages 3, 18, 20, 157, and 159.)

[NWS12] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. Patty: A taxonomy of relational patterns with semantic types. In *Proceedings of the Thirteenth Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12)*, 2012. (Cited on pages 11, 12, and 28.)

[NYT15] New York Times. http://www.nytimes.com/, 2015. [Online; accessed August 2015]. (Cited on page 127.)

[NZRS12a] Feng Niu, Ce Zhang, Christopher Ré, and Jude Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *International Journal on Semantic Web and Information Systems*, 8(3):42–73, 2012. (Cited on page 20.)

[NZRS12b] Feng Niu, Ce Zhang, Christopher Ré, and Jude W Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. 2012. (Cited on pages 21, 22, 207, and 222.)

[ODP15] Open Directory Project. http://www.dmoz.org/, 2015. [Online; accessed August 2015]. (Cited on pages 64 and 109.)

[oJC97] United States. Dept. of Justice and United States. Federal Trade Commission. *Horizontal merger guidelines*. U.S. Dept. of Justice, 1997. (Cited on page 65.)

[ON10] Christopher Olston and Marc Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010. (Cited on pages 14 and 15.)

[Ope15a] Open Calais: Bring structure to unstructured content. http://new.opencalais.com/, 2015. [Online; accessed August 2015]. (Cited on pages 67 and 179.)

[ope15b] OpenNLP: Java machine learning toolkit for natural language processing. http://opennlp.apache.org/, 2015. [Online; accessed August 2015]. (Cited on pages 29 and 139.)

[OWB09] Philip V. Ogren, Philipp G. Wetzler, and Steven J. Bethard. ClearTK: A framework for statistical natural language processing. In *UIMA Workshop at GSCL (UIMA@GSCL '09)*, 2009. (Cited on pages 29 and 30.)

[PDB13] Aditya Pal, Nilesh N. Dalvi, and Kedar Bellare. Discovering hierarchical structure for sources and entities. In *Proceedings of the Twenty-seventh National Conference on Artificial Intelligence (AAAI '13)*, 2013. (Cited on page 220.)

[Pea00] Karl Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, 50(302):157–175, 1900. (Cited on page 70.)

[Pen55] Roger Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(03):406–413, 1955. (Cited on page 173.)

[Pen56] Roger Penrose. On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, 1956. (Cited on page 173.)

[PR07] Siddharth Patwardhan and Ellen Riloff. Effective information extraction with semantic affinity patterns and relevant regions. In *Proceedings of the Eighth Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '07)*, 2007. (Cited on page 159.)

[PRH04] Patrick Pantel, Deepak Ravichandran, and Eduard Hovy. Towards terascale knowledge acquisition. In *Proceedings of the Twentieth international conference on Computational Linguistics (COLING '04)*, 2004. (Cited on page 222.)

[PRK93] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the Twenty-Seventh Annual Asilomar Conference on Signals, Systems, and Computers (ASILOMAR '93)*, 1993. (Cited on page 164.)

[PS07] Simone Paolo Ponzetto and Michael Strube. Deriving a large scale taxonomy from wikipedia. In *Proceedings of the Twenty-second National Conference on Artificial Intelligence (AAAI '07)*, 2007. (Cited on page 22.)

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Fifteenth Conference on Empirical Methods in Natural Language Processing (EMNLP '14)*, 2014. (Cited on pages 166 and 180.)

[Pub15] PubMed: US national library of medicine national institutes of health. http://www.ncbi.nlm.nih.gov/pubmed, 2015. [Online; accessed August 2015]. (Cited on pages 15 and 91.)

[PY13] Jakub Piskorski and Roman Yangarber. Information extraction: Past, present and future. In *Multi-source, Multilingual Information Extraction and Summarization*. 2013. (Cited on page 208.)

[Qia13] Richard Qian. Satori: Understand your world with bing. https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/, 2013. [Online; accessed August 2015]. (Cited on page 22.)

[RF08] Volker Roth and Bernd Fischer. The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms. In *Proceedings of*

the Twenty-Fifth International Conference on Machine Learning (ICML '08), 2008. (Cited on page 163.)

[RSJ12] Karthik Raman, Pannaga Shivaswamy, and Thorsten Joachims. Online learning to diversify from implicit feedback. In *Proceedings of the Eighteenth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD '12)*, 2012. (Cited on page 161.)

[RSo15] The R project for statistical computing. http://www.r-project.org/, 2015. [Online; accessed August 2015]. (Cited on page 114.)

[San08] Evan Sandhaus. The New York Times Annotated Corpus. In *Linguistic Data Consortium*, 2008. (Cited on pages 14, 137, and 178.)

[Sar08] Sunita Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008. (Cited on pages 9 and 14.)

[SBM96] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the Nineteenth ACM International Conference on Research and Development in Information Retrieval (SIGIR '96)*, 1996. (Cited on page 174.)

[SC03] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the Twenty-sixth ACM International Conference on Research and Development in Information Retrieval (SIGIR '03)*, 2003. (Cited on pages 98, 105, and 205.)

[SC04] Luo Si and Jamie Callan. Unified utility maximization framework for resource selection. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management (CIKM '04)*, 2004. (Cited on pages 93, 95, and 205.)

[SC05] Luo Si and Jamie Callan. Modeling search engine effectiveness for federated search. In *Proceedings of the Twenty-eighth ACM International*

Conference on Research and Development in Information Retrieval (SIGIR '05), 2005. (Cited on page 205.)

[SCAC14] Rodrygo L.T. Santos, Pablo Castells, Ismail Sengor Altingovde, and Fazli Can. Diversity and novelty in web search, recommender systems and data streams. In *Proceedings of the Seventh ACM International Conference on Web Search and Data Mining (WSDM '14)*, 2014. (Cited on pages 161 and 217.)

[Scu09] David Sculley. Large scale learning to rank. In *Advances in Ranking Workshop at NIPS (NIPS '09)*, 2009. (Cited on pages 130 and 134.)

[SDNR07] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *Proceedings of the Thirty-third International Conference on Very Large Databases (VLDB '07)*, 2007. (Cited on page 206.)

[SFMB07] Horacio Saggion, Adam Funk, Diana Maynard, and Kalina Bontcheva. Ontology-based information extraction for business intelligence. In *Proceedings of The Semantic Web, Sixth International Semantic Web Conference, Second Asian Semantic Web Conference (ISWC '07 / ASWC '07)*, 2007. (Cited on page 23.)

[SGD11] Balaji R. Soundrarajan, Thomas Ginter, and Scott L. DuVall. An interface for rapid natural language processing development in UIMA. In *Proceedings of the Fourty-nineth Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations (HLT '11)*, 2011. (Cited on page 28.)

[SGG13] Gonçalo Simões, Helena Galhardas, and Luis Gravano. When speed has a price: Fast information extraction using approximate algorithms. In *Proceedings of the Thirty-ninth International Conference on Very Large Databases (VLDB '13)*, 2013. (Cited on pages 21, 126, 205, 206, and 223.)

[SH12] Robert Speer and Catherine Havasi. Representing general relational knowledge in ConceptNet 5. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC '12)*, 2012. (Cited on page 22.)

[She09] Denis Shestakov. On building a search interface discovery system. In *Proceedings of the Second International Workshop on Resource Discovery (RED '09)*, 2009. (Cited on page 65.)

[Shl05] Jonathon Shlens. A tutorial on Principal Component Analysis. In *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*, 2005. (Cited on pages 168 and 181.)

[Sho07] Milad Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. In *Proceedings of the Twenty-ninth European conference on Information Retrieval research (ECIR '07)*, 2007. (Cited on page 205.)

[Sin12] Amit Singhal. Introducing the Knowledge Graph: things, not strings. http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html, 2012. [Online; accessed August 2015]. (Cited on page 22.)

[SIW06] Fabian M. Suchanek, Georgiana Ifrim, and Gerhard Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In *Proceedings of the Twelfth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD '06)*, 2006. (Cited on page 10.)

[SJCO02] Luo Si, Rong Jin, Jamie Callan, and Paul Ogilvie. A language modeling framework for resource selection and results merging. In *Proceedings of the Eleventh ACM International Conference on Information and Knowledge Management (CIKM '02)*, 2002. (Cited on page 205.)

[SK09] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4:2–4:2, 2009. (Cited on page 221.)

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A core of semantic knowledge. In *Proceedings of the Sixteenth International Conference on World Wide Web (WWW '07)*, 2007. (Cited on pages 10, 22, and 208.)

[SMK09] Ashwin Swaminathan, Cherian V. Mathew, and Darko Kirovski. Essential pages. In *Proceedings of the 2009 International Conference on Web Intelligence (WI '09)*, 2009. (Cited on page 161.)

[SMO10] Rodrygo L.T. Santos, Craig Macdonald, and Iadh Ounis. Selectively diversifying web search results. In *Proceedings of the Nineteenth ACM International Conference on Information and Knowledge Management (CIKM '10)*, 2010. (Cited on page 161.)

[Sod99] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999. (Cited on page 28.)

[Soe14] David Soergel. JLibSVM: Efficient training of Support Vector Machines in Java. http://dev.davidsoergel.com/trac/jlibsvm/, 2014. [Online; accessed August 2015]. (Cited on page 40.)

[SOM10] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: Real-time event detection by social sensors. In *Proceedings of the Nineteenth International Conference on World Wide Web (WWW '10)*, 2010. (Cited on page 209.)

[SP14] Fabian M. Suchanek and Nicoleta Preda. Semantic culturomics (vision paper). *VLDB Endowment*, 7(12):1215–1218, 2014. (Cited on page 24.)

[SRG10] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. Learning optimally diverse rankings over large document collections. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML '10)*, 2010. (Cited on page 161.)

[SS11] Milad Shokouhi and Luo Si. Federated search. *Foundations and Trends in Information Retrieval*, 5(1):1–102, 2011. (Cited on pages 19, 91, 93, 203, and 205.)

[SSSS07] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML '07)*, 2007. (Cited on pages 7, 125, 131, 141, and 182.)

[SSW03] Carl-Erik Särndal, Bengt Swensson, and Jan Wretman. *Model Assisted Survey Sampling*. Springer, 2003. (Cited on page 204.)

[sta15a] Stanford CoreNLP suit. http://nlp.stanford.edu/software/, 2015. [Online; accessed August 2015]. (Cited on pages 29 and 110.)

[Sta15b] Stanford named-entity recognizer. http://nlp.stanford.edu/software/CRF-NER.shtml, 2015. [Online; accessed August 2015]. (Cited on pages 33, 67, 139, and 179.)

[Sto74] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36(2):111–147, 1974. (Cited on page 42.)

[Str93] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1993. (Cited on page 177.)

[SZ07] Milad Shokouhi and Justin Zobel. Federated text retrieval from uncooperative overlapped collections. In *Proceedings of the Thirtieth ACM International Conference on Research and Development in Information Retrieval (SIGIR '07)*, 2007. (Cited on pages 93 and 95.)

[TAC06] Jordi Turmo, Alicia Ageno, and Neus Català. Adaptive information extraction. *ACM Computing Surveys*, 38(2):4, 2006. (Cited on page 10.)

[TBC+15] Nina Tahmasebi, Lars Borin, Gabriele Capannini, Devdatt Dubhashi, Peter Exner, Markus Forsberg, Gerhard Gossen, FredrikD. Johansson, Richard Johansson, Mikael Kågebäck, Olof Mogren, Pierre Nugues, and Thomas Risse. Visions and open challenges for a knowledge-based culturomics. *International Journal on Digital Libraries*, 15(2-4):169–187, 2015. (Cited on page 24.)

[Tik15] Apache Tika: A content analysis toolkit. http://tika.apache.org, 2015. [Online; accessed August 2015]. (Cited on page 65.)

[TM11] Oscar Täckström and Ryan T. McDonald. Discovering fine-grained sentiment with latent variable structured prediction models. In *Proceedings of the Thirty-Third European conference on Information Retrieval research (ECIR '11)*, 2011. (Cited on page 162.)

[TPL10] Domonkos Tikk, Peter Palaga, and Ulf Leser. A fast and effective dependency graph kernel for PPI relation extraction. *BMC Bioinformatics*, 11(S-5):P8, 2010. (Cited on page 179.)

[TRE00] Text REtrieval Conference. http://trec.nist.gov/, 2000. [Online; accessed August 2015]. (Cited on pages 2, 66, 109, 138, 159, and 179.)

[TS06] Andrew Turpin and Falk Scholer. User performance versus precision measures for simple search tasks. In *Proceedings of the Twenty-ninth ACM International Conference on Research and Development in Information Retrieval (SIGIR '06)*, 2006. (Cited on page 141.)

[TS09] Paul Thomas and Milad Shokouhi. SUSHI: Scoring scaled samples for server selection. In *Proceedings of the Thirty-second ACM International Conference on Research and Development in Information Retrieval (SIGIR '09)*, 2009. (Cited on page 206.)

[TSW06] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. TopX and XXL at INEX 2005. In *Proceedings of the Fourth International Conference on Initiative for the Evaluation of XML Retrieval (INEX '06)*, 2006. (Cited on page 22.)

[TT05] Yoshimasa Tsuruoka and Jun'ichi Tsujii. Chunk parsing revisited. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT '05)*, 2005. (Cited on page 13.)

[TTA09] Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *Proceedings of the Forty-seventh Annual Meeting of the Association for Computational Linguistics and the Fourth International Joint Conference on Natural Language Processing of the AFNLP (ACL '09)*, 2009. (Cited on page 132.)

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience: Research articles. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005. (Cited on page 222.)

[VC11] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*, 2011. (Cited on pages 161 and 218.)

[vdL10] Mark P. J. van der Loo. Distribution based outlier detection for univariate data. Technical Report Discussion paper 10003, Statistics Netherlands, The Hague, 2010. (Cited on page 114.)

[VGJL95] Ellen M. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the Eighteenth ACM International Conference on Research and Development in Information Retrieval (SIGIR '95)*, 1995. (Cited on page 206.)

[VHL10] Theresa Velden, Asif-ul Haque, and Carl Lagoze. A new approach to analyzing patterns of collaboration in co-authorship networks: mesoscopic analysis and interpretation. *Scientometrics*, 85(1):219–242, 2010. (Cited on page 24.)

[VHM+11] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *VLDB Endowment*, 4(9):528–538, 2011. (Cited on page 208.)

[VHSP10] Sarah Vieweg, Amanda L. Hughes, Kate Starbird, and Leysia Palen. Microblogging during two natural hazards events: What twitter may contribute to situational awareness. In *Proceedings of the Twenty-eighth Annual Conference on Human Factors in Computing Systems (CHI '10)*, 2010. (Cited on page 209.)

[Wal06] Christopher Walker. ACE 2005 multilingual training corpus. In *Linguistic Data Consortium*, 2006. (Cited on pages 33 and 45.)

[Whi09] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009. (Cited on pages 20 and 222.)

[Wik15] Wikipedia: The free encyclopedia. https://www.wikipedia.org/, 2015. [Online; accessed August 2015]. (Cited on pages 14 and 166.)

[Win99] William E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999. (Cited on page 18.)

[WKPU08] Casey Whitelaw, Alex Kehlenbeck, Nemanja Petrovic, and Lyle Ungar. Web-scale named entity recognition. In *Proceedings of the Seventeenth ACM International Conference on Information and Knowledge Management (CIKM '08)*, 2008. (Cited on page 138.)

[WL03] Jiying Wang and Frederick H. Lochovsky. Data extraction and label assignment for web databases. In *Proceedings of the Twelfth International Conference on World Wide Web (WWW '03)*, 2003. (Cited on page 65.)

[WSCN08] Richard C. Wang, Nico Schlaefer, William W. Cohen, and Eric Nyberg. Automatic set expansion for list question answering. In *Proceedings of the Seventh Conference on Empirical Methods in Natural Language Processing (EMNLP '08)*, 2008. (Cited on page 23.)

[WSE13] Henning Wachsmuth, Benno Stein, and Gregor Engels. Information extraction as a filtering task. In *Proceedings of the Twenty-second ACM International Conference on Information and Knowledge Management (CIKM '13)*, 2013. (Cited on pages 3, 20, and 159.)

[XC98] Jinxi Xu and Jamie Callan. Effective retrieval with distributed collections. In *Proceedings of the Twenty-first ACM International Conference on Research and Development in Information Retrieval (SIGIR '98)*, 1998. (Cited on page 205.)

[XC99] Jinxi Xu and William Bruce Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the Twenty-second ACM International Conference on Research and Development in Information Retrieval (SIGIR '99)*, 1999. (Cited on page 205.)

[XGZ11] Wei Xu, Ralph Grishman, and Le Zhao. Passage retrieval for information extraction using distant supervision. In *Proceedings of the Fifth International Joint Conference on Natural Language Processing (IJCNLP '11)*, 2011. (Cited on pages 3, 4, 20, and 160.)

[Yat34] Frank Yates. Contingency tables involving small numbers and the chi-square test. *Supplement to the Journal of the Royal Statistical Society*, 1(2):217–235, 1934. (Cited on page 70.)

[YC04] Hui Yang and Tat-Seng Chua. Web-based list question answering. In *Proceedings of the Twentieth international conference on Computational Linguistics (COLING '04)*, 2004. (Cited on page 23.)

[YCB+07] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: Open information extraction on the web. In *Proceedings of Human Language Technologies Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '07)*, 2007. (Cited on page 207.)

[YHO+11] Dani Yogatama, Michael Heilman, Brendan O'Connor, Chris Dyer, Bryan R. Routledge, and Noah A. Smith. Predicting a scientific community's response to an article. In *Proceedings of the Twelfth Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*, 2011. (Cited on pages 15, 24, and 219.)

[YJ08] Yisong Yue and Thorsten Joachims. Predicting diverse subsets using structural svms. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML '08)*, 2008. (Cited on page 161.)

[YL97] Budi Yuwono and Dik Lun Lee. Server ranking for distributed text retrieval systems on the internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA '97)*, 1997. (Cited on page 205.)

[YL06] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, 68(1):49–67, 2006. (Cited on page 163.)

[YPWC+13] Huahai Yang, Daina Pupons-Wickham, Laura Chiticariu, Yunyao Li, Benjamin Nguyen, and Arnaldo Carreno-Fuentes. I can do text analytics!: Designing development tools for novice developers. In *Proceedings*

of the Thirty-first Annual Conference on Human Factors in Computing Systems (CHI '13), 2013. (Cited on page 28.)

[YS14] Dani Yogatama and Noah A. Smith. Making the most of bag of words: Sentence regularization with alternating direction method of multipliers. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning (ICML '14)*, 2014. (Cited on page 162.)

[YXKL10] Haiqin Yang, Zenglin Xu, Irwin King, and Michael R. Lyu. Online learning for group lasso. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML '10)*, 2010. (Cited on page 163.)

[YYC10] Ainur Yessenalina, Yisong Yue, and Claire Cardie. Multi-level structured models for document-level sentiment classification. In *Proceedings of the Eleventh Conference on Empirical Methods in Natural Language Processing (EMNLP '10)*, 2010. (Cited on page 162.)

[ZAR+03] Dmitry Zelenko, Chinatsu Aone, Anthony Richardella, Jaz, Thomas Hofmann, Tomaso Poggio, and John Shawe-taylor. Kernel methods for relation extraction. *The Journal of Machine Learning Research*, 3:1083–1106, 2003. (Cited on pages 11, 12, and 28.)

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the Second USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10)*, 2010. (Cited on page 20.)

[ZCL03] Cheng Xiang Zhai, William W. Cohen, and John Lafferty. Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *Proceedings of the Twenty-sixth ACM International Conference on Research and Development in Information Retrieval (SIGIR '03)*, 2003. (Cited on page 161.)

[ZG05] Shubin Zhao and Ralph Grishman. Extracting relations with integrated information using kernel methods. In *Proceedings of the Forty-third An-*

nual Meeting on Association for Computational Linguistics (ACL '05), 2005. (Cited on pages 11, 12, and 28.)

[ZH05] Hui Zou and Trevor Hastie. Regularization and variable selection via the Elastic Net. *Journal of the Royal Statistical Society*, 67(2):301–320, 2005. (Cited on pages 132, 163, and 182.)

[Zil08] Marcus P. Zillman. Deep web research 2008. http://www.llrx.com/features/deepweb2008.htm, 2008. [Online; accessed August 2015]. (Cited on page 15.)

[ZWC+13] Qinghua Zheng, Zhaohui Wu, Xiaocheng Cheng, Lu Jiang, and Jun Liu. Learning to crawl deep web. *Information Systems*, 38(6):801–819, 2013. (Cited on page 15.)

[ZWS04] Zhaohui Zheng, Xiaoyun Wu, and Rohini Srihari. Feature selection for text categorization on imbalanced data. *SIGKDD Explorations Newsletter*, 6(1):80–89, 2004. (Cited on page 111.)

[ZZD11] Mingyang Zhang, Nan Zhang, and Gautam Das. Mining a search engine's corpus: Efficient yet unbiased sampling and aggregate estimation. In *Proceedings of the 2011 ACM International Conference on Management of Data (SIGMOD '11)*, 2011. (Cited on pages 16, 53, 94, 95, 99, 101, 105, 203, and 204.)

[ZZD13] Mingyang Zhang, Nan Zhang, and Gautam Das. Mining a search engine's corpus without a query pool. In *Proceedings of the Twenty-second ACM International Conference on Information and Knowledge Management (CIKM '13)*, 2013. (Cited on pages 16, 94, 95, 96, 102, 103, 107, 108, 113, 203, 204, and 218.)